MATLAB®
para ingenieros

Holly Moore

PEARSON
Prentice Hall®

MATLAB® para ingenieros

MATLAB® para ingenieros

Holly Moore
Salt Lake Community College
Salt Lake City, Utah

TRADUCCIÓN
Víctor Campos Olguín
Traductor profesional

REVISIÓN TÉCNICA
Rogelio Márquez Nuño
Escuela Superior de Ingeniería Química e Industrias Extractivas
Instituto Politécnico Nacional



México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador
España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay • Venezuela

Datos de catalogación bibliográfica

Moore, Holly

MATLAB® para ingenieros. Primera edición

PEARSON EDUCACIÓN, México, 2007

ISBN: 978-970-26-1082-3
Área: Ingeniería

Formato: 20 × 25.5 cm Páginas: 624

Edición en español

Editor:	Luis Miguel Cruz Castillo e-mail: luis.cruz@pearsoned.com
Editor de desarrollo:	Felipe Hernández Carrasco
Supervisor de producción:	Enrique Trejo Hernández

Edición en inglés

Editorial Director, ECS:	Marcia J. Horton
Senior Editor:	Holly Stark
Associate Editor:	Dee Bernhard
Editorial Assistant:	Nicole Kunzmann
Executive Managing Editor:	Vince O'Brien
Managing Editor:	David A. George
Production Editor:	Scott Disanno
Director of Creative Services:	Paul Belfanti
Art Director:	Jonathan Boylan
Cover Designer:	Bruce Kenselaar
Art Editor:	Greg Dulles
Manufacturing Manager:	Alexis Heydt-Long
Manufacturing Buyer:	Lisa McDowell

PRIMERA EDICIÓN, 2007

ISBN 10: 970-26-1082-6
ISBN 13: 978-970-26-1082-3

Impreso en México. *Printed in Mexico.*
1 2 3 4 5 6 7 8 9 0 - 10 09 08 07



Contenido

Revisores de ESource

Agradecemos a todos los que nos ayudaron o revisaron textos de esta serie.

Naeem Abdurrahman, *University of Texas, Austin*
Stephen Allan, *Utah State University*
Anil Bajaj, *Purdue University*
Grant Baker, *University of Alaska–Anchorage*
William Beckwith, *Clemson University*
Haym Benaroya, *Rutgers University*
John Biddle, *California State Polytechnic University*
Tom Bledsaw, *ITT Technical Institute*
Fred Boadu, *Duk University*
Tom Bryson, *University of Missouri, Rolla*
Ramzi Bualuan, *University of Notre Dame*
Dan Budny, *Purdue University*
Betty Burr, *University of Houston*
Joel Cahoon, *Montana State University*
Dale Calkins, *University of Washington*
Linda Chattin, *Arizona State University*
Harish Cherukuri, *University of North Carolina–Charlotte*
Arthur Clausing, *University of Illinois*
Barry Crittendon, *Virginia Polytechnic and State University*
Donald Dabdub, *University of CA Irvine*
Kurt DeGoede, *Elizabethtown College*
John Demel, *Ohio State University*
James Devine, *University of South Florida*
Heidi A. Diefes-Dux, *Purdue University*
Jerry Dunn, *Texas Tech University*
Ron Eaglin, *University of Central Florida*
Dale Elifrits, *University of Missouri, Rolla*
Christopher Fields, *Drexel University*
Patrick Fitzhorn, *Colorado State University*
Susan Freeman, *Northeastern University*
Howard M. Fulmer, *Villanova University*
Frank Gerlitz, *Washtenaw Community College*
John Glover, *University of Houston*
John Graham, *University of North Carolina–Charlotte*
Ashish Gupta, *SUNY at Buffalo*
Otto Gygax, *Oregon State University*
Malcom Heimer, *Florida International University*
Donald Herling, *Oregon State University*
Thomas Hill, *SUNY at Buffalo*
A. S. Hodel, *Auburn University*
Kathryn Holliday-Darr, *Penn State U Behrend College, Erie*
Tom Horton, *University of Virginia*
James N. Jensen, *SUNY at Buffalo*
Mary Johnson, *Texas A & M Commerce*
Vern Johnson, *University of Arizona*
Jean C. Malzahn Kampe, *Virginia Polytechnic Institute and State University*
Autar Kaw, *University of South Florida*
Kathleen Kitto, *Western Washington University*
Kenneth Klika, *University of Akron*
Harold Knickle, *University of Rhode Island*
Terry L. Kohutek, *Texas A&M University*
Bill Leahy, *Georgia Institute of Technology*
John Lumkes, *Purdue University*
Mary C. Lynch, *University of Florida*
Melvin J. Maron, *University of Louisville*
James Mitchell, *Drexel University*
Robert Montgomery, *Purdue University*
Nikos Mourtos, *San Jose State University*
Mark Nagurka, *Marquette University*
Romarathnam Narasimhan, *University of Miami*
Shahnam Navee, *Georgia Southern University*
James D. Nelson, *Louisiana Tech University*
Soronadi Nnaji, *Florida A&M University*
Sheila O'Connor, *Wichita State University*
Kevin Passino, *Ohio State University*
Ted Pawlicki, *University of Rochester*
Ernesto Penado, *Northern Arizona University*
Michael Peshkin, *Northwestern University*
Ralph Pike, *Louisiana State University*
Matt Ohland, *Clemson University*
Dr. John Ray, *University of Memphis*
Stanley Reeves, *Auburn University*
Larry Richards, *University of Virginia*
Marc H. Richman, *Brown University*
Christopher Rowe, *Vanderbilt University*
Liz Rozell, *Bakersfield College*
Heshem Shaalem, *Georgia Southern University*
Tabb Schreder, *University of Toledo*

Randy Shih, *Oregon Institute of Technology*
Howard Silver, *Fairleigh Dickenson University*
Avi Singhal, *Arizona State University*
Tim Sykes, *Houston Community College*
Toby Teorey, *University of Michigan*
Neil R. Thompson, *University of Waterloo*
Raman Menon Unnikrishnan, *Rochester Institute of Technology*
Michael S. Wells, *Tennessee Tech University*
Ed Wheeler, *University of Tennessee at Martin*
Joseph Wujek, *University of California, Berkeley*
Edward Young, University of South Carolina
Garry Young, Oklahoma State University
Steve Yurgartis, Clarkson University
Mandochehr Zoghi, University of Dayton

Acerca de este libro

Este libro se desarrolló conforme enseñaba MATLAB y otros lenguajes de computación a estudiantes nuevos de ingeniería en la Salt Lake Community College. Me sentía frustrada por la falta de un texto que "comenzara desde el principio". Aunque había muchos libros de referencia muy abarcadores, todos suponían un nivel elevado de matemáticas y conocimientos de computación que mis estudiantes no tenían. Además, debido a que MATLAB fue adoptado en sus orígenes por profesionales de los campos de procesamiento de señales e ingeniería eléctrica, la mayoría de dichos textos daban ejemplos principalmente de dichas áreas, enfoque que no se adaptaba a un currículo general de ingeniería. Este texto comienza con álgebra básica y muestra cómo se puede usar MATLAB para resolver problemas de ingeniería de un rango amplio de disciplinas. Los ejemplos proceden de conceptos que se introducen en las primeras clases de química y física, y de las clases de ingeniería de los primeros semestres. Se utiliza de manera consistente una metodología estándar para resolver problemas.

El libro supone que el estudiante tiene una comprensión básica del álgebra universitaria y ha recibido una introducción a conceptos trigonométricos; los estudiantes más avanzados en matemáticas, por lo general, hacen progresos más rápidos a través del material. Aunque el texto no pretende enseñar materias como estadística o álgebra matricial, se incluye una descripción breve cuando se estudian las técnicas de MATLAB relacionadas con tales materias. Además, las secciones que describen las técnicas de MATLAB para resolver problemas mediante cálculo y ecuaciones diferenciales se introducen casi al final de los capítulos apropiados. Esas secciones se pueden destinar para un estudio adicional a los estudiantes con formación más avanzada de matemáticas, o pueden ser útiles como material de referencia conforme los estudiantes avanzan en su carrera de ingeniería.

El libro pretende ser un manual "manos a la obra". Mis estudiantes tienen más éxito cuando leen el libro sentados frente a una computadora y escriben los ejemplos conforme surgen. En el texto se incluyen numerosos ejemplos, y en cada capítulo se incluyen ejemplos numerados más complicados para reforzar los conceptos que se introducen. Los ejercicios de práctica de cada capítulo dan a los estudiantes la oportunidad inmediata de usar sus nuevas habilidades, y en el Apéndice B se dan las soluciones completas.

El material se agrupa en tres secciones. La primera, ***Introducción a las herramientas básicas de MATLAB***, es la iniciación del estudiante y contiene los capítulos siguientes:

- El capítulo 1 muestra cómo se usa MATLAB en la ingeniería e introduce una metodología estándar de resolución de problemas.
- El capítulo 2 introduce el ambiente de MATLAB y las habilidades que se requieren para ejecutar cálculos básicos. Este capítulo también introduce los archivos-m (*m-files*). Al hacerlo tan pronto en el texto hace más fácil que los estudiantes guarden su trabajo y desarrollen una estrategia de programación consistente.
- El capítulo 3 detalla la gran variedad de problemas que se pueden resolver con las funciones internas de MATLAB. El material de apoyo de muchas de las funciones se proporciona para ayudar al estudiante a comprender cómo se pueden usar. Por ejemplo, se describe la diferencia entre números aleatorios gaussianos y uniformes, y se presentan ejemplos de cada uno.
- En el capítulo 4 se demuestra el poder de la formulación de problemas en MATLAB mediante el uso de matrices y se abunda sobre las técnicas empleadas para definir dichas matrices. En este capítulo se introduce la función *meshgrid* y se emplea para resolver problemas con dos variables. En el capítulo 5 se estudia de nuevo el difícil concepto de las variables mezcladas (*meshing*), cuando se introducen las gráficas de superficies.
- El capítulo 5 describe la gran variedad de técnicas de graficación bidimensional y tridimensional disponible en MATLAB. Se hace énfasis en la creación de gráficas por medio de los comandos de MATLAB, ya sea desde la ventana de comandos o desde un archivo-m. Sin embargo, también se introducen las extremadamente valiosas técnicas de edición interactiva de gráficas y de creación de gráficas directamente desde la ventana de trabajo.

MATLAB es un poderoso lenguaje de programación que incluye los conceptos básicos comunes a la mayoría de los lenguajes de programación. Puesto que se trata de un lenguaje con base en *scripts*, la creación de programas y su depuración en MATLAB con frecuencia es más fácil que en los lenguajes tradicionales de programación, como C++. Esto hace de MATLAB una valiosa herramienta para las clases introductorias a la programación. La segunda sección del texto, ***Programación en MATLAB***, introduce a los estudiantes a la programación y consta de los capítulos siguientes:

- El capítulo 6 describe cómo crear y usar funciones definidas por el usuario. Este capítulo también enseña a los estudiantes cómo crear una "caja de herramientas" de funciones para usar en sus propios proyectos de programación.
- El capítulo 7 introduce funciones que interactúan con el usuario del programa, incluidas entradas definidas por el usuario, la salida formateada y técnicas gráficas de entrada. Este capítulo también introduce el modo de celdas para crear archivos-m y describe las numerosas funciones de entrada y salida que permiten que MATLAB importe datos desde diversos formatos de archivo.
- El capítulo 8 describe las funciones lógicas y muestra cómo crear código MATLAB con estructuras de control (*for*, *while* e *if*). Se enfatiza el uso de las funciones lógicas sobre las estructuras de control, en parte porque los estudiantes (y maestros) con experiencia de programación previa con frecuencia pasan por alto las ventajas de usar la funcionalidad matricial interna de MATLAB.

Los capítulos del 1 al 8 se deben enseñar en forma secuencial, pero los capítulos de la sección 3, ***Conceptos avanzados de MATLAB***, no dependen uno del otro. Cualquiera o todos ellos podrían usarse en un curso introductorio o servir como material de referencia para el autoaprendizaje. La mayor parte del material es apropiado para principiantes. Un curso de dos créditos podría incluir los capítulos del 1 al 8 y el capítulo 9, mientras que un curso de tres créditos incluiría los 13 capítulos, pero eliminaría las secciones 11.4, 11.5, 12.4, 12.5 y 12.6, que describen técnicas de diferenciación, de integración y de solución de ecuaciones diferenciales. Las habilidades que se desarrollan en los siguientes capítulos serán especialmente útiles conforme los estudiantes se involucren más en la solución de problemas de ingeniería:

- El capítulo 9 estudia la solución de problemas con álgebra matricial, inclusive los productos punto y cruz y la solución de sistemas lineales de ecuaciones. Aunque el álgebra de matrices

se emplea mucho en todos los campos de la ingeniería, encuentra aplicación temprana en las clases de estadística y dinámica que se imparten en la mayoría de las especialidades de ingeniería.

- El capítulo 10 es una introducción a la gran variedad de tipos de datos disponibles en MATLAB. Este capítulo es especialmente útil para los estudiantes de ingeniería eléctrica y de computación.
- El capítulo 11 introduce el paquete de matemáticas simbólicas de MATLAB, construido sobre el motor Maple 8. Los estudiantes descubrirán que este material es especialmente valioso en sus clases de matemáticas. Mis estudiantes me dicen que el paquete es uno de los conjuntos de técnicas más valiosos que se introducen en el curso. Es algo que comienzan a usar de inmediato.
- El capítulo 12 presenta técnicas numéricas que se emplean en una amplia variedad de aplicaciones, en especial el ajuste de curvas y la estadística. Los estudiantes valoran estas técnicas cuando toman clases de laboratorio tales como química o física, o cuando cursan laboratorios asociados con clases de ingeniería, como transferencia de calor, dinámica de fluidos o resistencia de materiales.
- El capítulo 13 examina técnicas gráficas que se emplean para visualizar datos. Estas técnicas son especialmente útiles para analizar los resultados de los cálculos de análisis numérico, inclusive los del análisis estructural, dinámica de fluidos y códigos de transferencia de calor.

El apéndice A lista todas las funciones y símbolos (o caracteres) especiales que se introducen en el texto. El apéndice B contiene las soluciones completas de todos los ejercicios de práctica. El sitio www.pearsoneducacion.net/moore, exclusivo para el profesor, incluye el material siguiente:

- Archivos-m con las soluciones de los ejercicios de práctica.
- Archivos-m que contienen las soluciones de los problemas de ejemplo.
- Archivos-m con las respuestas a los problemas de tarea.
- Transparencias de PowerPoint para cada capítulo.
- Todas las figuras que se emplean en el texto, adecuadas para incluirlas en sus propias presentaciones de PowerPoint.

DEDICATORIA Y AGRADECIMIENTOS

Este proyecto no habría sido posible sin el apoyo de mi familia, que soportó la lectura de múltiples borradores del texto y comió muchas pizzas congeladas mientras me concentraba en la escritura. Gracias a Mike, Heidi, Meagan, Dave y Vinnie, y a mi esposo, el doctor Steven Purcell.

Este libro lo dedico a mi padre, el profesor George Moore, quien impartió cátedra durante casi 20 años en el Departamento de Ingeniería Eléctrica en la South Dakota School of Mines and Technology. El profesor Moore obtuvo su título profesional a los 54 años de edad, después de una exitosa carrera como piloto de la Fuerza Aérea de los Estados Unidos, y es un recordatorio viviente de que nunca se es demasiado viejo para aprender.



CAPÍTULO

1

Acerca de MATLAB

Objetivos

Después de leer este capítulo, el alumno será capaz de

- entender qué es MATLAB y por qué se usa ampliamente en ingeniería y ciencia.
- comprender las ventajas y limitaciones de la edición estudiantil de MATLAB.
- formular problemas mediante el uso de un enfoque estructurado de resolución de problemas.

1.1 ¿QUÉ ES MATLAB?

MATLAB es una de las muchas sofisticadas herramientas de computación disponibles en el comercio para resolver problemas de matemáticas, tales como Maple, Mathematica y MathCad. A pesar de lo que afirman sus defensores, ninguna de ellas es "la mejor". Todas tienen fortalezas y debilidades. Cada una permitirá efectuar cálculos matemáticos básicos, pero difieren en el modo como manejan los cálculos simbólicos y procesos matemáticos más complicados, como la manipulación de matrices. Por ejemplo, MATLAB es superior en los cálculos que involucran matrices, mientras que Maple lo supera en los cálculos simbólicos. El nombre mismo de MATLAB es una abreviatura de **Mat**rix **Lab**oratory, laboratorio matricial. En un nivel fundamental, se puede pensar que estos programas son sofisticadas calculadoras con base en una computadora. Son capaces de realizar las mismas funciones que una calculadora científica, y **muchas más**. Si usted tiene una computadora en su escritorio, descubrirá que usará MATLAB en lugar de su calculadora incluso para la más simple de sus aplicaciones matemáticas, por ejemplo para el balance de su chequera. En muchas clases de ingeniería, la realización de cálculos con un programa de computación matemático como MATLAB sustituye la programación de computadoras más tradicional. Esto no significa que el lector no deba aprender un lenguaje de alto nivel como C++ o FORTRAN, sino que los programas como MATLAB se han convertido en una herramienta estándar para ingenieros y científicos.

Dado que MATLAB es tan fácil de usar, muchas tareas de programación se llevan a cabo con él. Sin embargo, MATLAB no siempre es la mejor herramienta para usar en una tarea de programación. El programa destaca en cálculos numéricos, especialmente en los relacionados con matrices y gráficas, pero usted no querrá escribir un programa de procesamiento de palabras en MATLAB. C++ y FORTRAN son programas de propósito general y serían los programas de elección para aplicaciones grandes como los sistemas operativos o el software de diseño. (De hecho, MATLAB, que *es* un programa grande de aplicación, se escribió originalmente en FORTRAN y después se rescribió en C, precursor de C++.) Por lo general, los programas de alto nivel no ofrecen acceso fácil a la graficación, que es una aplicación en la que destaca MATLAB. El área principal de interferencia entre MATLAB y los programas de alto nivel es el "procesamiento de números": programas que requieren cálculos repetitivos o el procesamiento de grandes cantidades de datos. Tanto MATLAB como los programas de alto nivel son buenos en el procesamiento de números. Por lo general, es más fácil escribir un programa que "pro-

Idea clave: MATLAB es óptimo para cálculos matriciales.

cese números" en MATLAB, pero usualmente se ejecutará más rápido en C++ o FORTRAN. La única excepción a esta regla son los cálculos que involucran matrices: puesto que MATLAB es óptimo para matrices, si un problema se puede formular con una solución matricial, MATLAB lo ejecuta sustancialmente más rápido que un programa similar en un lenguaje de alto nivel.

MATLAB está disponible en versiones tanto profesional como estudiantil. Es probable que en el laboratorio de cómputo de su colegio o universidad esté instalada la versión profesional, pero disfrutará tener la versión estudiantil en casa. MATLAB se actualiza de manera regular; este texto se basa en MATLAB 7. Si utiliza MATLAB 6 podrá observar algunas diferencias menores entre éste y MATLAB 7. En versiones anteriores a MATLAB 5.5 existen diferencias sustanciales.

1.2 EDICIÓN ESTUDIANTIL DE MATLAB

Idea clave: MATLAB se actualiza regularmente.

Las ediciones profesional y estudiantil de MATLAB son muy similares. Es probable que los estudiantes que comienzan no sean capaces de distinguir la diferencia. Las ediciones estudiantiles están disponibles para los sistemas operativos Microsoft Windows, Mac OSX y Linux, y se pueden adquirir en las librerías escolares o en línea a través de The MathWorks, en www.mathworks.com.

MathWorks empaca su software en grupos llamados entregas (*releases)*, y MATLAB 7 se agrupa, junto con otros productos, como Simulink 6.1, en la entrega 14. El número de entrega es el mismo para ambas ediciones, estudiantil y profesional. La entrega 14 de la edición estudiantil incluye las siguientes características:

- MATLAB 7 completo.
- Simulink 6.1, con la capacidad de construir modelos de hasta 1000 bloques (la versión profesional permite un número ilimitado de bloques).
- Grandes porciones de Symbolic Math Toolbox.
- Manuales de software tanto para MATLAB 7 como para Simulink.
- Un CD que contiene la documentación electrónica completa.
- Una licencia de usuario único, que en el caso de los estudiantes se limita al empleo en el salón de clase (la licencia de la versión profesional es tanto individual como grupal).

Cajas de herramientas distintas a la Symbolic Math Toolbox se pueden adquirir por separado.

La diferencia más grande que observará entre las ediciones profesional y estudiantil es el incitador de comando (prompt), que es

```
>>
```

en la versión profesional, y es

```
EDU>>
```

en la versión estudiantil.

1.3 ¿CÓMO SE USA MATLAB EN LA INDUSTRIA?

La habilidad para usar herramientas tales como MATLAB se convirtió rápidamente en un requisito para muchos puestos de ingeniería. En una reciente búsqueda de empleo en Monster.com se encontró el siguiente anuncio:

> ... se busca un ingeniero de sistema de pruebas con experiencia en aviónica... Sus responsabilidades incluyen modificación de scripts de MATLAB, ejecución de simulaciones en Simulink y el análisis de los datos del resultado. El candidato DEBE estar familiarizado con MATLAB, Simulink y C++...

Este anuncio no es raro. La misma búsqueda arrojó 75 compañías diferentes que requerían específicamente el manejo de MATLAB para los ingenieros que entraran al nivel de base.

MATLAB es particularmente popular para aplicaciones de ingeniería eléctrica, aunque se usa muchísimo en todos los campos de la ingeniería y ciencias. Las secciones que siguen delinean sólo algunas de las muchas aplicaciones actuales que utilizan MATLAB.

Idea clave: MATLAB se usa ampliamente en ingeniería.

1.3.1 Ingeniería eléctrica

MATLAB se utiliza mucho en ingeniería eléctrica para aplicaciones de procesamiento de señales. Por ejemplo, en la figura 1.1 se presentan varias imágenes creadas durante un programa de investigación en la University of Utah para simular algoritmos de detección de colisiones que usan las moscas domésticas (y adaptados en el laboratorio a sensores de silicio). La investigación dio como resultado el diseño y fabricación de un chip de computadora que detecta colisiones inminentes. Esto tiene una aplicación potencial en el diseño de robots autónomos que usen la visión para navegar y en particular en aplicaciones para la seguridad en automóviles.

1.3.2 Ingeniería biomédica

Por lo general, las imágenes médicas se guardan como archivos dicom (el estándar *Digital Imaging and Communications in Medicine*: imágenes digitales y comunicaciones en medicina). Los archivos dicom utilizan la extensión de archivo .dcm. La compañía MathWorks ofrece una caja de herramientas adicional, llamada caja de herramientas para imágenes que puede leer esos archivos, lo que hace que sus datos estén disponibles para procesamiento en MATLAB. La caja de herramientas para imágenes también incluye un amplio rango de funciones de las que muchas son especialmente apropiadas para las imágenes médicas. Un conjunto limitado de datos MRI ya convertidos a un formato compatible con MATLAB se incluye con el programa MATLAB estándar. Este conjunto de datos le permite probar algunas de las funciones de generación de imágenes disponibles tanto con la instalación estándar de MATLAB como con la caja de herramientas para imágenes expandida, si la tiene instalada en su computadora. La figura 1.2 muestra seis imágenes de secciones horizontales del cerebro con base en el conjunto de datos MRI.

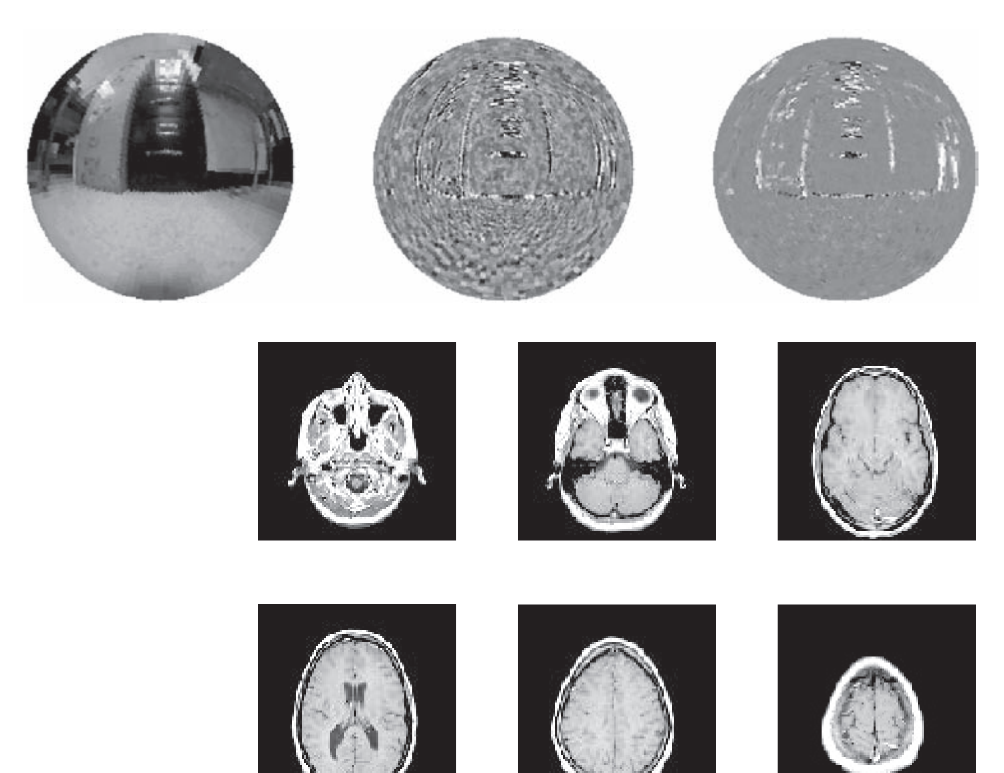


Figura 1.1
Procesamiento de imágenes con el uso de una cámara con objetivo de ojo de pescado para simular el sistema visual del cerebro de una mosca doméstica. (Con permiso del Dr. Reid Harrison, University of Utah.)

Figura 1.2
Secciones horizontales del cerebro, con base en el archivo de datos de muestra incluido con MATLAB.

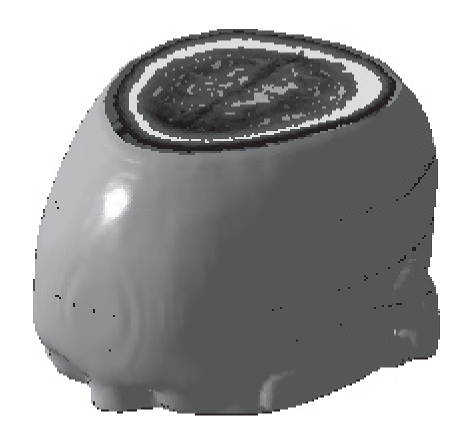
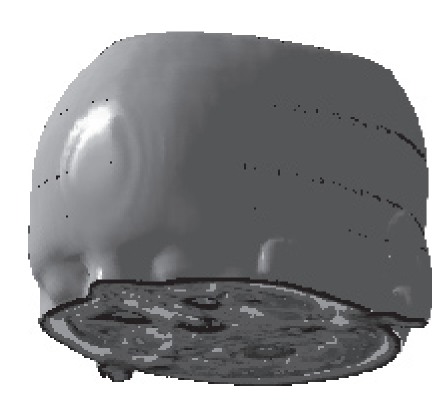



Figura 1.3
Visualización tridimensional de datos MRI.

El mismo conjunto de datos se puede usar para construir una imagen tridimensional, como cualquiera de las que se muestran en la figura 1.3. En el tutorial **help** se dan instrucciones detalladas acerca de cómo crear esas imágenes.

1.3.3 Dinámica de fluidos

Los cálculos que describen velocidades de fluidos (rapideces y direcciones) son importantes en varios campos. En particular, a los ingenieros aeroespaciales les interesa el comportamiento de los gases, tanto afuera de una aeronave o vehículo espacial como dentro de las cámaras de combustión. Visualizar el comportamiento tridimensional de los fluidos es difícil, pero MATLAB ofrece cierto número de herramientas que lo hacen más sencillo. En la figura 1.4, los resultados del cálculo de campo de flujo para un dispositivo de control del vector de empuje se representan como una gráfica de vectores de velocidad. El control del vector de empuje es el proceso de cambiar la dirección en que apunta una tobera (y, por tanto, la dirección en que se mueve un cohete) al operar un actuador (un dispositivo pistón-cilindro). El modelo en la figura

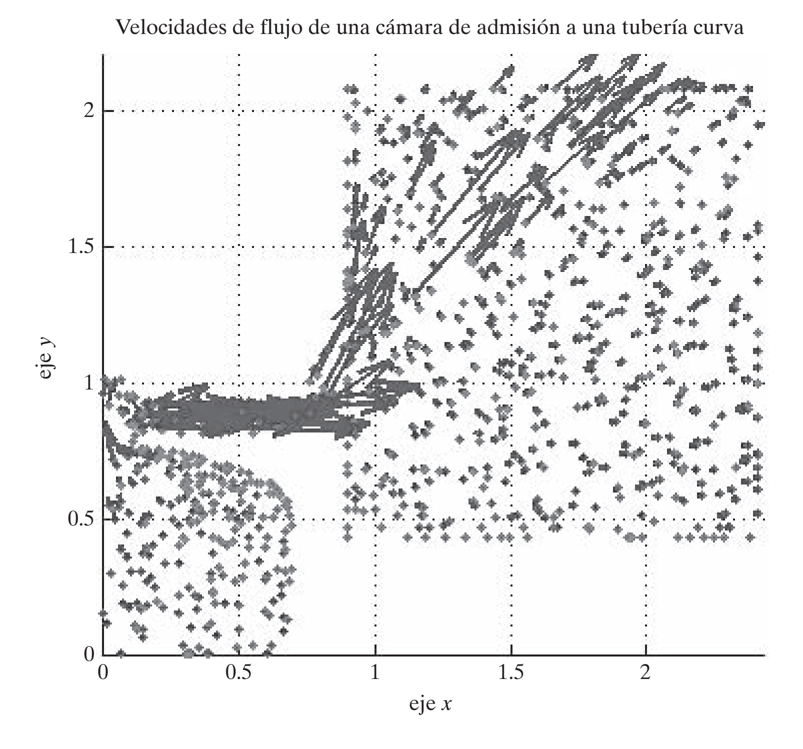


Figura 1.4
Gráfica de vectores de velocidad del comportamiento de un gas en un dispositivo de control del vector de empuje.

representa un depósito de gas a alta presión (una cámara de admisión) que eventualmente se alimenta al pistón y, por tanto, controla la longitud del actuador.

1.4 RESOLUCIÓN DE PROBLEMAS EN INGENIERÍA Y CIENCIAS

En las disciplinas de ingeniería, ciencias y programación de computadoras, es importante tener un enfoque consistente para resolver los problemas técnicos. El enfoque que se plantea a continuación es útil en cursos tan distintos como química, física, termodinámica y diseño de ingeniería. También se aplica a las ciencias sociales, como economía y sociología. Otros autores quizá formulen sus esquemas de resolución de problemas de forma ligeramente diferente, pero todos tienen el mismo formato básico:

Idea clave: use siempre una estrategia sistemática de resolución de problemas.

- **Plantear el problema.**
  - En esta etapa con frecuencia es útil hacer un dibujo.
  - Si no tiene una comprensión clara del problema, es improbable que pueda resolverlo.
- **Describir** los valores de **entrada** (conocidos) y las **salidas** (incógnitas) que se requieren.
  - Tenga cuidado de incluir las unidades conforme describe los valores de entrada y salida. El manejo descuidado de las unidades con frecuencia lleva a respuestas incorrectas.
  - Identifique las constantes que tal vez requiera en el cálculo, como la constante de los gases ideales y la aceleración de la gravedad.
  - Si es apropiado, en un dibujo escriba los valores que haya identificado o agrúpelos en una tabla.
- Desarrollar un **algoritmo** para resolver el problema. En aplicaciones de cómputo, es frecuente que esto se logre con una **prueba de escritorio**. Para ello necesitará.
  - Identificar cualesquiera ecuaciones que relacionen los valores conocidos con las incógnitas.
  - Trabajar con una versión simplificada del problema, a mano o con calculadora.
- **Resolver** el problema. En este libro, esta etapa involucra la creación de una **solución con MATLAB**.
- **Probar la solución**.
  - ¿Sus resultados tienen sentido físico?
  - ¿Coinciden con los cálculos de la muestra?
  - ¿La respuesta es la que se pedía en realidad?
  - Las gráficas con frecuencia son formas útiles de verificar que los cálculos son razonables.

Si utiliza en forma consistente un enfoque estructurado de resolución de problemas, como el que se acaba de describir, descubrirá que los problemas tipo “narración” son mucho más fáciles de resolver. El ejemplo 1.1 ilustra esta estrategia de resolución de problemas.

EJEMPLO 1.1

La conversión de la materia en energía

Albert Einstein (véase la figura 1.5) es con toda seguridad el físico más famoso del siglo veinte. Einstein nació en Alemania en 1879 y asistió a la escuela tanto en Alemania como en Suiza. Mientras trabajaba en una oficina de patentes en Berna desarrolló su famosa teoría de la relatividad. Acaso la ecuación física mejor conocida en la actualidad es su:

$$E = mc^2$$

Esta ecuación sorprendentemente sencilla vincula los mundos antes separados de la materia y la energía, y se puede utilizar para encontrar la cantidad de energía que se libera conforme la materia se destruye en reacciones nucleares tanto naturales como inducidas por el hombre.

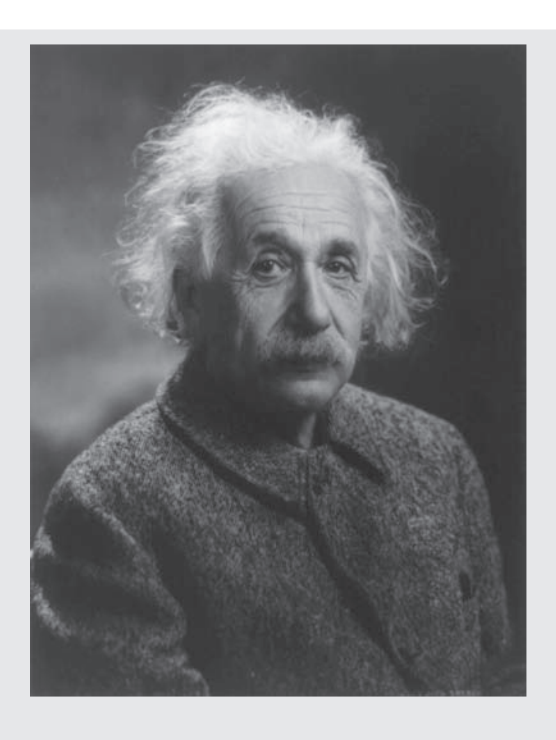


Figura 1.5
Albert Einstein. (Cortesía de la Biblioteca del Congreso, LC-USZ62-60242.)

El Sol irradia 385×10^{24} J/s de energía, los cuales se generan mediante reacciones nucleares que convierten la materia en energía. Utilice MATLAB y la ecuación de Einstein para determinar cuánta materia se debe convertir en energía para producir esa cantidad de radiación en un día.

1. Plantee el problema.
   Encontrar la cantidad de materia que se necesita para producir la cantidad de energía que irradia el Sol cada día
2. Describa la entrada y la salida.

   Entrada

   Energía $E = 385 \times 10^{24}$ J/s, que se debe convertir en la energía total irradiada durante un día
   Rapidez de la luz $c = 3.0 \times 10^8$ m/s

   Salida

   Masa m en kg
3. Desarrolle una prueba de escritorio.
   La energía irradiada en un día es

$$385 \times 10^{24} \frac{\text{J}}{\text{s}} \times 3600 \frac{\text{s}}{\text{hora}} \times 24 \frac{\text{horas}}{\text{día}} \times 1 \text{ día} = 3.33 \times 10^{31} \text{ J}$$

   La ecuación $E = mc^2$ se debe resolver para m y sustituir los valores de E y c. Se tiene

$$m = E/c^2$$

$$m = \frac{3.33 \times 10^{31} \text{ J}}{(3.0 \times 10^8 \text{ m/s})^2}$$

$$= 3.7 \times 10^{14} \frac{\text{J}}{\text{m}^2/\text{s}^2}$$

A partir de los criterios de salida se puede ver que se desea la masa en kg, así que, ¿qué fue lo que estuvo mal? Se necesita hacer una conversión de unidades adicional:

$$1 \text{ J} = 1 \text{ kg m}^2/\text{s}^2$$

$$= 3.7 \times 10^{14} \frac{\text{kg m}^2/\text{s}^2}{\text{m}^2/\text{s}^2} = 3.7 \times 10^{14} \text{ kg}$$

4. Desarrolle una solución con MATLAB.
   Claramente, en este punto de su estudio de MATLAB no ha aprendido cómo crear código MATLAB. Sin embargo, debe ser capaz de ver, a partir de la siguiente muestra de código, que la sintaxis de MATLAB es similar a la sintaxis que se utiliza en la mayoría de las calculadoras científicas algebraicas. Los comandos de MATLAB se introducen en el prompt (**>>**), y los resultados se reportan en la línea siguiente. El código es:

```
>> E=385e24
E =
  3.8500e+026
>> E=E*3600*24
E =
  3.3264e+031
>> c=3e8
c =
  300000000
>> m=E/c^2
m =
  3.6960e+014
```

   De aquí en adelante no se mostrará el prompt cuando se describan interacciones en la ventana de comandos.
5. Pruebe la solución.
   La solución MATLAB coincide con el cálculo hecho a mano, ¿pero tienen sentido los números? Cualquier cosa multiplicada por 10^{14} es un número realmente grande. Sin embargo, considere que la masa del Sol es 2×10^{30} kg. Se podría calcular cuánto tiempo tomará consumir por completo la masa del Sol a una tasa de 3.7×10^{14} kg/día. Se tiene:

$$\text{tiempo} = (\text{masa del Sol})/(\text{tasa de consumo})$$

$$\text{tiempo} = \frac{2 \times 10^{30} \text{ kg}}{3.7 \times 10^{14} \text{ kg/día}} \times \frac{\text{año}}{365 \text{ días}} = 1.5 \times 10^{13} \text{ años}$$

   ¡Esto es 15 billones de años! En lo que le resta de vida, no necesitará preocuparse de que el Sol se quede sin materia que pueda convertir en energía.



CAPÍTULO 2

Ambiente MATLAB

Objetivos

Después de leer este capítulo, el alumno será capaz de

- iniciar el programa MATLAB y resolver problemas simples en la ventana de comando.
- comprender el uso de matrices que hace MATLAB.
- identificar y usar las diversas ventanas de MATLAB.
- definir y usar matrices simples.
- nombrar y usar variables.
- entender el orden de operaciones en MATLAB.
- comprender la diferencia entre cálculos escalares, de arreglo y matriciales en MATLAB.
- expresar números en notación de punto flotante y científica.
- ajustar el formato que se usa para desplegar números en la ventana de comandos.
- guardar el valor de las variables que se usen en una sesión de MATLAB.
- guardar una serie de comandos en un archivo-m.

2.1 INICIO

Usar MATLAB por primera vez es fácil; dominarlo puede tomar años. En este capítulo se introducirá al lector al ambiente de MATLAB y se le mostrará cómo efectuar cálculos matemáticos básicos. Después de leer este capítulo, será capaz de empezar a usar MATLAB para hacer sus tareas o en el trabajo. Por supuesto, conforme complete el resto de los capítulos podrá hacer más cosas.

Dado que el procedimiento para instalar MATLAB depende de su sistema operativo y del ambiente de la computadora, se supondrá que el lector ya instaló MATLAB en su computadora o que trabaja en un laboratorio de computación donde ya se instaló MATLAB. Para iniciar MATLAB, ya sea en el ambiente Windows o en el de Apple, haga clic en el icono del escritorio, o use el menú inicio para encontrar el programa. En el ambiente UNIX, escriba **Matlab** en el shell prompt (línea de comandos). No importa cómo lo inicie, una vez abierto MATLAB debe ver el prompt (incitador) de MATLAB (**>>** o **EDU>>**), que le indica que MATLAB está listo para que ingrese un comando. Cuando termine su sesión con MATLAB, puede salir del programa al escribir **quit** o **exit** en el prompt de MATLAB. MATLAB también utiliza la barra de menú estándar de Windows, de modo que puede salir del programa si elige **EXIT MATLAB** en el menú File (Archivo) o al seleccionar el icono de cerrar (**x**) en la esquina superior derecha de la pantalla. En la figura 2.1 se muestra por defecto la pantalla de MATLAB, la cual se abre siempre que inicia el programa.

Para comenzar a usar MATLAB, sólo necesita prestar atención a la ventana de comandos (a la derecha de la pantalla). En ella puede realizar cálculos en forma similar a como lo hace en una calculadora científica. Incluso, la mayoría de la sintaxis es la misma. Por ejemplo, para calcular el valor de 5 al cuadrado, se escribe el comando

```
5^2
```

Se desplegará la salida siguiente:

```
ans =
        25
```

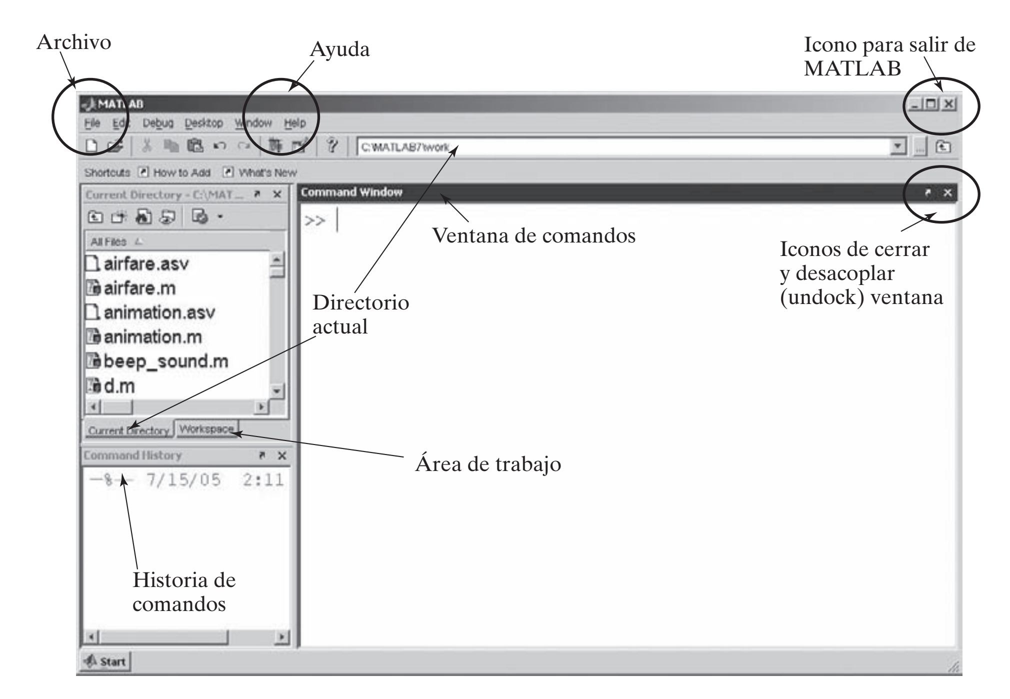


Figura 2.1
Ventana de apertura de MATLAB. El ambiente de MATLAB consta de algunas ventanas, cuatro de las cuales se abren en la vista por defecto. Otras se abren conforme se necesiten durante una sesión de MATLAB.

O bien, para encontrar el valor de cos(π), escriba

```
cos(pi)
```

que da como resultado la salida

```
ans =
      -1
```

Idea clave: MATLAB usa las reglas algebraicas estándar para el orden de operación.

MATLAB usa las reglas algebraicas estándar para ordenar operaciones, lo que se vuelve importante cuando encadena cálculos. Estas reglas se estudian en la sección 2.3.2.

Sugerencia

Es posible que piense que algunos de los ejemplos son demasiado sencillos como para escribirlos usted mismo, que es suficiente con leer el material. Sin embargo, ¡recordará mejor el material si lo lee y lo escribe!

Antes de continuar, intente resolver el ejercicio de práctica 2.1.

Ejercicio de práctica 2.1

Escriba las siguientes expresiones en el prompt de comando de MATLAB y observe los resultados:

1. 5+2
2. 5*2
3. 5/2

4. 3 + 2*(4 + 3)
5. 2.54*8 / 2.6
6. 6.3 − 2.1045
7. 3.6^2
8. 1 + 2^2
9. sqrt(5)
10. cos(pi)

Sugerencia

Es posible que encuentre frustrante descubrir que, cuando comete un error, no se puede volver a escribir el comando después de haberlo ejecutado. Esto ocurre porque la ventana de comandos crea una lista de todos los comandos que ingresó. No se puede "desejecutar" o "descrear" un comando. Lo que puede hacer es introducir el comando en forma correcta y luego ejecutar esta nueva versión. MATLAB le ofrece varias maneras de facilitar lo anterior. Una de ellas es usar las teclas de flecha, que, por lo general, se encuentran en el lado derecho del teclado. La tecla hacia arriba, ↑, le permite desplazarse a través de la lista de comandos ejecutados. Una vez que encuentre el comando apropiado, puede editarlo y luego ejecutar la versión nueva. Esto ahorra tiempo en verdad. Sin embargo, también siempre es posible sólo volver a escribir el comando.

2.2 VENTANAS DE MATLAB

MATLAB utiliza varias ventanas de despliegue. La vista por defecto, que se presenta en la figura 2.1, incluye una gran ***command window*** (ventana de comandos) a la derecha y, apiladas a la izquierda, se encuentran las ventanas ***current directory*** (directorio actual), ***workspace*** (área de trabajo) y ***command history*** (historia de comandos). Observe las pestañas abajo a la izquierda de las ventanas; dichas pestañas le permiten acceder a las ventanas ocultas. Las versiones antiguas de MATLAB también incluían una ventana de ***launch pad*** (lanzamiento), que se reemplazó con el botón de ***start*** (inicio) en la esquina inferior izquierda. Además, cuando sea necesario, se abrirán automáticamente ventanas de ***document*** (documento), ***graphics*** (gráficas) y ***editing*** (edición). Cada una de esas ventanas se describe en las secciones que siguen. MATLAB también tiene construida internamente una función de ayuda, a la que se puede acceder desde la barra de menú, como se muestra en la figura 2.1. Para personalizar el escritorio, puede redimensionar cualquiera de estas ventanas, cerrar las que no use con el icono de cerrar (la x en la esquina superior derecha de cada ventana), o "desacoplarlas" con el icono undock (desacoplar), ⇲, que también se localiza en la esquina superior derecha de cada ventana.

2.2.1 Ventana de comandos (command window)

La ventana de comandos se localiza en el lado derecho de la vista por defecto de la pantalla de MATLAB, como se muestra en la figura 2.1. La ventana de comandos ofrece un ambiente similar a una memoria de trabajo auxiliar (scratch pad). El empleo de la ventana de comandos le permite guardar los valores que calcule, mas no los ***comandos*** que usó para generarlos. Si desea guardar la secuencia de comandos, necesitará emplear la ventana de edición para crear un **archivo-m** (m-file). Los archivos-m se describen en la sección 2.4.2. Ambos enfoques son valiosos; sin embargo, primero se enfatizará el uso de la ventana de comandos, antes de introducir los archivos-m.

Idea clave: la ventana de comandos es similar a una memoria de trabajo auxiliar.

2.2.2 Historia de comandos (command history)

Idea clave: la historia de comandos registra todos los comandos que se escribieron en la ventana de comandos.

La ventana de ***historia de comandos*** registra los comandos que se escriben en la ventana de comandos. Cuando sale de MATLAB, o cuando escribe el comando **clc**, la ventana de comandos se limpia (clear). Sin embargo, la ventana de historia de comandos conserva una lista de todos sus comandos. También puede limpiar la historia de comandos con el menú *edit*. Si trabaja en una computadora pública, entonces, como medida de seguridad, las opciones de MATLAB por defecto se pueden establecer de modo que limpie la historia cuando salga del programa. Si introdujo los comandos de muestra anteriores, observará que se repiten en la ventana de historia de comandos. Esta ventana es valiosa por varias razones, dos de las cuales son: porque permite revisar sesiones anteriores de MATLAB y porque se puede usar para transferir comandos a la ventana de comandos. Por ejemplo, primero limpie el contenido de la ventana de comandos al escribir

```
clc
```

Esta acción limpia la ventana de comandos, pero deja intactos los datos de la ventana de historia de comandos. Usted puede transferir cualquier comando desde la ventana de historia de comandos hacia la ventana de comandos al hacer doble clic (lo que también ejecuta el comando) o al hacer clic y arrastrar la línea de código a la ventana de comandos. Intente hacer doble clic

```
cos(pi)
```

en la ventana de historia de comandos. Debe regresar

```
ans =
      -1
```

Ahora haga clic y arrastre

```
5^2
```

desde la ventana de historia de comando hacia la ventana de comandos. El comando no se ejecutará hasta que oprima *enter*, y entonces obtendrá el resultado:

```
ans =
      25
```

Conforme ejecute cálculos cada vez más complicados en la ventana de comandos, encontrará que la ventana de historia de comandos es útil.

2.2.3 Ventana del área de trabajo (workspace)

Idea clave: la ventana workspace (área de trabajo) lista la información que describe todas las variables que crea el programa.

La ***ventana del área de trabajo*** le mantiene informado de las ***variables*** que usted define conforme ejecuta comandos en la ventana de comandos. Si ha hecho los ejemplos, la ventana del área de trabajo debe mostrar sólo una variable, **ans**, y decir que tiene un valor de 25 y que es un arreglo doble:

Name	Value	Class
⊞ ans	25	double array

Haga que la ventana del área de trabajo diga algo más acerca de esta variable al hacer clic con el botón derecho sobre la barra con las etiquetas de las columnas. (Esta característica es nueva en MATLAB 7 y no funcionará con una versión anterior.) Revise **size** (tamaño) y **bytes**, además de **name** (nombre), **value** (valor) y **class** (clase). La ventana del área de trabajo ahora debe mostrar la siguiente información:

Name	Value	Size	Bytes	Class
⊞ **ans**	**25**	**1 × 1**	**8**	**double array**

El símbolo en forma de retícula indica que la variable **ans** es un arreglo. El tamaño, 1 × 1, dice que es un solo valor (una fila por una columna) y, por tanto, es un escalar. El arreglo usa 8 bytes de memoria. MATLAB está escrito en lenguaje C, y la designación de clase dice que, en lenguaje C, **ans** es un arreglo de punto flotante y doble precisión. En este momento, basta saber que la variable **ans** puede almacenar un número punto flotante (un número con punto decimal). En realidad, MATLAB considera a todo número que se ingrese como si fuera punto flotante, se escriban decimales o no.

Idea clave: el tipo de datos por defecto es de números punto flotante y doble precisión almacenados en una matriz.

Es posible definir variables adicionales en la ventana de comandos, y se listarán en la ventana del área de trabajo. Por ejemplo, al escribir

```
A = 5
```

regresa

```
A =
      5
```

Observe que la variable **A** se agregó a la ventana del área de trabajo, que lista las variables en orden alfabético. Las variables que comienzan con letras mayúsculas se listan en primer lugar, seguidas por las variables que comienzan con letras minúsculas.

Name	Value	Size	Bytes	Class
⊞ **A**	**5**	**1 × 1**	**8**	**double array**
⊞ **ans**	**25**	**1 × 1**	**8**	**double array**

En la sección 2.3.2 se estudiará en detalle cómo introducir matrices a MATLAB. Por ahora, usted puede ingresar una matriz unidimensional sencilla al escribir

```
B = [1, 2, 3, 4]
```

Este comando regresa

```
B =
      1     2     3     4
```

Las comas son opcionales; se obtendría el mismo resultado con

```
B = [ 1  2  3  4]
B =
      1     2     3     4
```

Observe que la variable **B** se agregó a la ventana del área de trabajo y que su tamaño es un arreglo 1 × 4:

Name	Value	Size	Bytes	Class
⊞ **A**	**5**	**1 × 1**	**8**	**double array**
⊞ **B**	**[1 2 3 4]**	**1 × 4**	**32**	**double array**
⊞ **ans**	**25**	**1 × 1**	**8**	**double array**

En forma similar se definen matrices bidimensionales. Se emplea punto y coma para separar las filas. Por ejemplo:

```
C = [ 1 2 3 4; 10 20 30 40; 5 10 15 20]
```

regresa

```
C =
        1     2     3     4
       10    20    30    40
        5    10    15    20
```

Name	Value	Size	Bytes	Class
A	5	1 × 1	8	double array
B	[1 2 3 4]	1 × 4	32	double array
C	<3 x 4 double>	3 × 4	96	double array
ans	25	1 × 1	8	double array

Observe que **C** aparece en la ventana del área de trabajo como una matriz de 3 × 4. Para conservar espacio, no se mencionan los valores almacenados en la matriz.

Se pueden recuperar los valores para cualquier variable al escribir en el nombre de la variable. Por ejemplo, al ingresar

```
A
```

se obtiene

```
A =
        5
```

Aunque las únicas variables que se introdujeron son matrices que contienen números, son posibles otros tipos de variables.

Al describir la ventana de comandos se introdujo el comando **clc**. Este comando limpia la ventana de comandos y deja una página en blanco para que usted trabaje en ella. Sin embargo, no borra de la memoria las variables reales que creó. El comando **clear** (limpiar) borra todas la variables guardadas. La acción del comando **clear** se refleja en la ventana del área de trabajo. Inténtelo al escribir

```
clear
```

en la ventana de comandos. La ventana del área de trabajo ahora está vacía:

Name	Value	Size	Bytes	Class

Si usted suprime la ventana del área de trabajo (al cerrarla o desde el menú file o con el icono de cierre en la esquina superior derecha de la ventana), todavía podrá descubrir cuáles variables se definieron mediante el comando **whos**:

```
whos
```

Si se hubiese ejecutado antes de ingresar el comando **clear**, **whos** habría regresado

Name	**Size**	**Bytes**	**Class**
A	**1x1**	**8**	**double array**
B	**1x4**	**32**	**double array**
C	**3x4**	**96**	**double array**
ans	**1x1**	**8**	**double array**

Grand total is 18 elements using 144 bytes

2.2.4 Ventana de directorio actual (current directory)

La ventana de directorio actual lista todos los archivos en una carpeta de la computadora llamada directorio actual. Cuando MATLAB ingresa a archivos o guarda información, usa el directorio actual a menos que se diga algo diferente. La ubicación por defecto del directorio actual varía con su versión del software y con cómo se instaló. Sin embargo, el directorio actual se cita en la parte superior de la ventana principal. El directorio actual se puede cambiar al seleccionar otro directorio de la lista desplegable que se ubica junto a la lista de directorio o al navegar entre los archivos de su computadora. La navegación se lleva a cabo con el botón browse, que se ubica junto a la lista desplegable. (Véase la figura 2.2.)

2.2.5 Ventana de documento (document window)

Hacer doble clic sobre cualquier variable mencionada en la ventana del área de trabajo lanza automáticamente una ventana de documento que contiene el **array editor** (editor de arreglos). Los valores que se almacenan en la variable se despliegan en un formato de hoja de cálculo. Puede cambiar los valores en el editor de arreglos o puede agregar nuevos valores. Por ejemplo, si todavía no ingresa la matriz bidimensional C, ingrese el siguiente comando en la ventana de comandos:

```
C = [ 1 2 3 4; 10 20 30 40; 5 10 15 20];
```

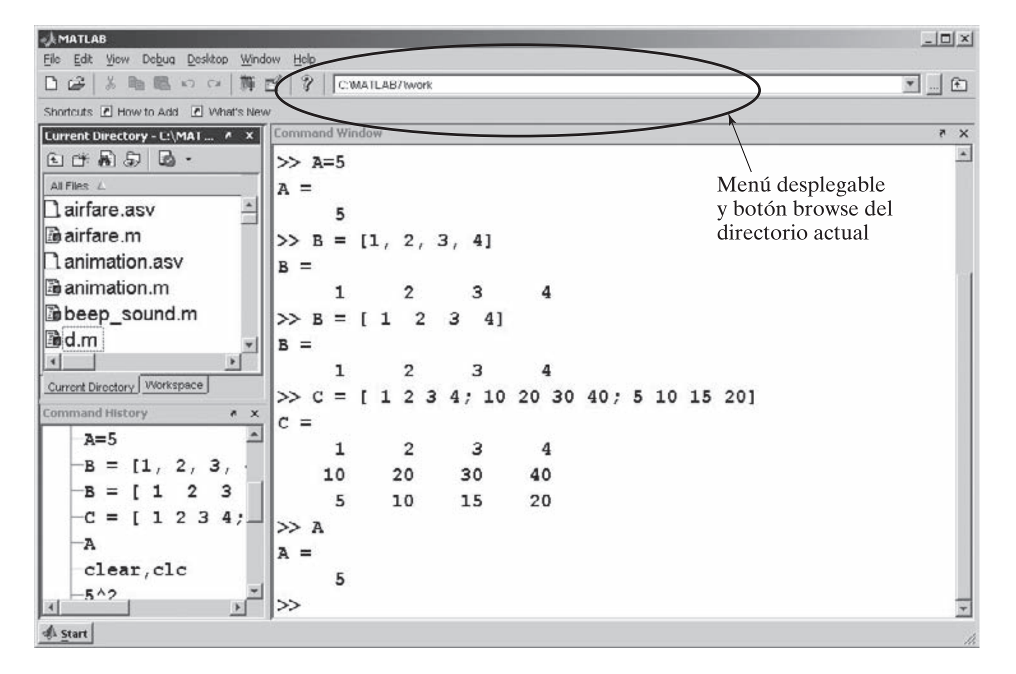


Figura 2.2
La ***ventana de directorio actual*** lista todos los archivos en el directorio actual. Puede cambiar el directorio actual o en el menú desplegable o con el botón browse.

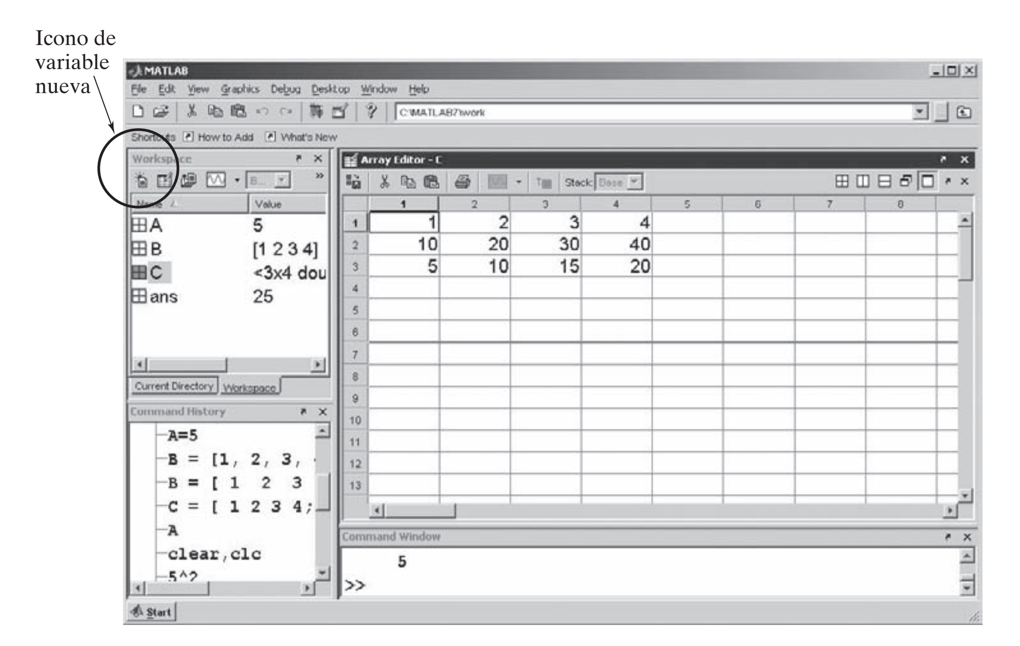


Figura 2.3
La ***ventana de documentos*** despliega el ***editor de arreglos***.

Idea clave: un punto y coma suprime la salida de los comandos escritos en la ventana de comandos.

Poner punto y coma al final del comando suprime la salida, de modo que no se repita en la ventana de comandos. Sin embargo, ahora **C** se debe citar en la ventana del área de trabajo. Haga doble clic en ella. Sobre la ventana de comandos se abrirá una ventana de documento, como se muestra en la figura 2.3. Ahora puede agregar más valores a la matriz **C** o cambiar los valores existentes.

La ventana de documento/editor de arreglos también se puede usar en conjunto con la ventana del área de trabajo para crear arreglos completamente nuevos. Corra su ratón lentamente sobre los iconos en la barra de atajos en lo alto de la ventana del área de trabajo. Si es paciente, deberá aparecer la función de cada icono. El icono de variable nueva se parece a una página con un gran asterisco detrás suyo. Seleccione el icono de variable nueva y, en la lista de variables, deberá aparecer una nueva variable llamada **unnamed** (sin nombrar). Puede cambiar su nombre al hacer clic derecho y seleccionar **rename** (renombrar) del menú secundario. Para agregar valores a esta variable nueva, haga doble clic sobre ella y agregue sus datos desde la ventana de editor de arreglo. El botón de variable nueva es una nueva característica de MATLAB 7; si utiliza una versión anterior, no podrá crear variables de esta forma.

Cuando termine de crear variables nuevas, cierre el editor de arreglos al seleccionar el icono de cerrar ventana en la esquina superior derecha de la ventana.

2.2.6 Ventana gráficas (graphics window)

La ventana de gráficas se lanza automáticamente cuando solicita una gráfica. Para demostrar esta característica, primero cree un arreglo de valores x:

```
x = [ 1 2 3 4 5];
```

(Recuerde que el punto y coma suprime la salida de este comando; sin embargo, en la ventana del área de trabajo aparece una variable nueva, x.)

Ahora cree una lista de valores y:

```
y = [10 20 30 40 50];
```

Para crear una gráfica, use el comando plot:

```
plot(x,y)
```

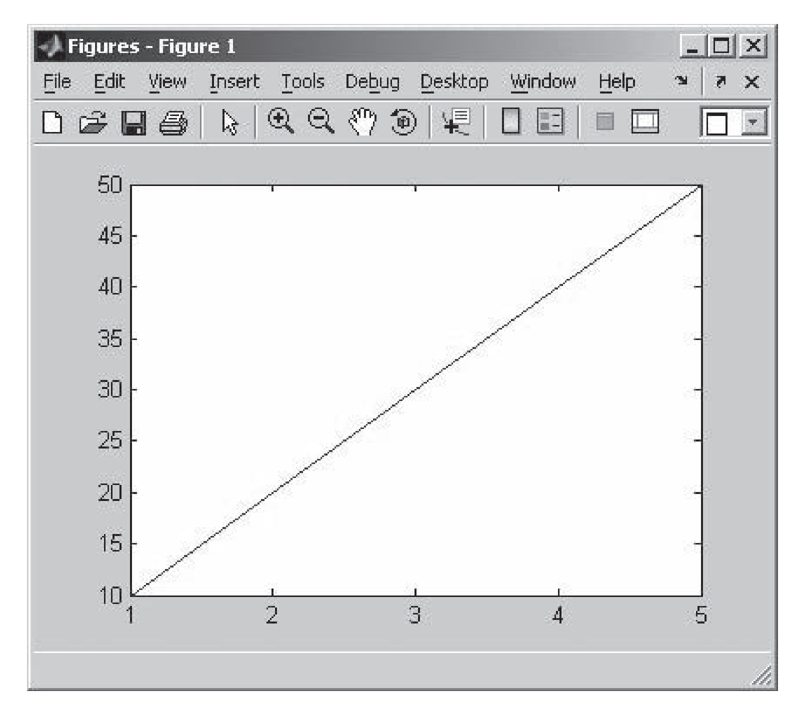


Figura 2.4
MATLAB hace fácil la creación de gráficas.

Idea clave: agregue siempre un título y etiquetas de ejes a las gráficas.

La ventana de gráficas se abre automáticamente. (Véase la figura 2.4.) Note que en la barra de tareas aparece una nueva etiqueta de ventana al fondo de la pantalla de la ventana. Se titulará **<Student Version> Figure...** o simplemente **Figure 1**, dependiendo de si usa la versión estudiantil o profesional, respectivamente, del software. Cualesquiera gráficas adicionales que cree sobrescribirán la figura 1 a menos que ordene específicamente a MATLAB que abra una nueva ventana de gráficas.

MATLAB facilita la modificación de las gráficas al agregar títulos, etiquetas *x* y *y*, líneas múltiples, etc. ¡Los ingenieros y científicos **nunca** presentan una gráfica sin etiquetas!

2.2.7 Ventana de edición (edit window)

La ventana de edición se abre al elegir **File** de la barra de menú, luego **New** y, finalmente, **M-file** (**File** → **New** → **M-file**). Esta ventana le permite escribir y guardar una serie de comandos sin ejecutarlos. También puede abrir la ventana de edición al escribir **edit** en el prompt de comando o al elegir el botón New File (archivo nuevo) en la barra de herramientas (toolbar).

2.2.8 Botón de inicio

El botón de inicio se ubica en la esquina inferior izquierda de la ventana de MATLAB. Ofrece un acceso alternativo a las diversas ventanas de MATLAB, así como a la función de ayuda, productos de Internet y cajas de herramientas de MATLAB. Las cajas de herramientas proporcionan funcionalidad MATLAB adicional, para áreas de contenido específico. En particular, la caja de herramientas simbólica es enormemente útil a científicos e ingenieros. El botón de inicio es nuevo a MATLAB 7 y sustituye la ventana de lanzamiento que se usaba en MATLAB 6.

2.3 RESOLUCIÓN DE PROBLEMAS CON MATLAB

El ambiente de ventana de comandos es una poderosa herramienta para resolver problemas de ingeniería. Para usarla de manera efectiva, necesitará entender más acerca de cómo funciona MATLAB.

2.3.1 Uso de variables

Aunque es posible resolver muchos problemas al usar MATLAB como calculadora, usualmente es más conveniente dar nombres a los valores que utiliza. MATLAB usa las convenciones de nomenclatura comunes a la mayoría de los programas de cómputo:

- Todos los nombres deben comenzar con una letra. Los nombres pueden tener cualquier longitud, pero en MATLAB 7 sólo se usan los primeros 63 caracteres. (Use el comando **namelengthmax** para confirmar esto cuando instale MATLAB.) Aunque MATLAB le permitirá crear nombres de variable largos, la longitud excesiva crea una significativa oportunidad de error. Un lineamiento común es usar letras minúsculas y números en los nombres de variable y usar letras mayúsculas para los nombres de constantes. Sin embargo, si una constante tradicionalmente se expresa como una letra minúscula, siéntase en libertad de seguir dicha convención. Por ejemplo, en los textos de física, la rapidez de la luz siempre es la letra *c* minúscula. Los nombres deben ser lo suficientemente cortos como para recordarlos y que sean descriptivos.
- Los únicos caracteres permisibles son letras, números y el guión bajo. Con el comando **isvarname** puede verificar si se permite el nombre de la variable. Como es estándar en los lenguajes de computación, el número 1 significa que algo es verdadero y el número 0 significa falso. En consecuencia,

```
isvarname time
ans =
      1
```

  indica que **time** es un nombre de variable legítimo, y

```
isvarname cool-beans
ans =
      0
```

  dice que **cool-beans** no es un nombre de variable legítimo.
- Los nombres son sensibles a mayúsculas/minúsculas. La variable **x** es diferente de la variable **X**.
- MATLAB reserva una lista de palabras clave para uso del programa, que no se pueden asignar como nombres de variable. El comando **iskeyword** hace que MATLAB elabore una lista de tales nombres reservados:

```
iskeyword
ans =
  'break'
  'case'
  'catch'
  'continue'
  'else'
  'elseif'
  'end'
  'for'
  'function'
  'global'
  'if'
  'otherwise'
  'persistent'
  'return'
  'switch'
  'try'
  'while'
```

- MATLAB le permite reasignar nombres de función internos como nombres de variable. Por ejemplo, podría crear una nueva variable llamada **sin** con el comando

```
sin=4
```

que regresa

```
sin =
     4
```

Esto es claramente una práctica peligrosa, pues la función **sin** (es decir: seno) ya no está disponible. Si intenta usar la función sobrescrita, obtendrá un enunciado de error (“Índice supera dimensiones de matriz”):

```
sin(3)
??? Index exceeds matrix dimensions.
```

Puede verificar si una variable es una función MATLAB interna al usar el comando **which**:

```
which sin
sin is a variable.
```

Puede restablecer **sin** a una función al escribir

```
clear sin
```

Ahora, cuando pregunte

```
which sin
```

la respuesta es

```
C:\MATLAB7\toolbox\matlab\elfun\sin.m
```

que menciona la ubicación de la función interna.

Ejercicio de práctica 2.2

¿Cuál de los siguientes nombres se permiten en MATLAB? Haga sus predicciones y luego pruébelas con los comandos **isvarname**, **iskeyword** y **which**.

1. test
2. Test
3. if
4. mi-libro
5. mi_libro
6. Esteesunnombremuylargoperoinclusoasisepermite?
7. 1ergrupo
8. grupo_uno
9. zzaAbc
10. z34wAwy?12#
11. sin
12. log

2.3.2 Matrices en MATLAB

Idea clave: la matriz es el tipo de datos principal en MATLAB y puede retener información numérica así como otros tipos de información.

El tipo de datos básico que se usa en MATLAB es la *matriz*. Un solo valor, llamado *escalar*, se representa como una matriz 1 × 1. Una lista de valores, ordenados o en una columna o en una fila, es una matriz unidimensional que se llama *vector*. Una tabla de valores se representa como una matriz bidimensional. Aunque este capítulo se limitará a escalares, vectores y matrices, MATLAB puede manejar arreglos de orden superior.

vector: matriz compuesta de una sola fila o una sola columna

En nomenclatura matemática, las matrices se representan como filas y columnas dentro de corchetes:

$$A = [5] \qquad B = [2 \quad 5] \qquad C = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$$

En este ejemplo, A es una matriz 1 × 1, B es una matriz 1 × 2 y C es una matriz 2 × 2. La ventaja de usar representación matricial es que todos los grupos de información se pueden representar con un solo nombre. La mayoría de personas se sienten más cómodas al asignar un nombre a un solo valor, así que se comenzará por explicar cómo MATLAB maneja los escalares y luego se avanzará a matrices más complicadas.

escalar: matriz de un solo valor

Operaciones escalares

MATLAB maneja operaciones aritméticas entre dos escalares en forma muy parecida a como lo hacen otros programas de cómputo e incluso su calculadora. En la tabla 2.1 se muestra la sintaxis para la suma, resta, multiplicación, división y exponenciación. El comando

```
a = 1 + 2
```

se debe leer como “a **a** se le asigna un valor de 1 más 2”, que es la suma de dos cantidades escalares. Las operaciones aritméticas entre dos variables escalares usa la misma sintaxis. Por ejemplo, suponga que usted definió **a** en el enunciado anterior y que **b** tiene un valor de 5:

```
b = 5
```

Entonces

```
x = a + b
```

regresa el siguiente resultado:

```
x =
     8
```

Tabla 2.1 Operaciones aritméticas entre dos escalares (operaciones binarias)

Operación	Sintaxis algebraica	Sintaxis MATLAB
Suma	$a + b$	**a + b**
Resta	$a - b$	**a – b**
Multiplicación	$a \times b$	**a* b**
División	$\frac{a}{b}$ o $a \div b$	**a / b**
Exponenciación	a^b	**a^b**

En MATLAB, un solo signo igual (=) se llama operador asignación. El operador asignación hace que el resultado de sus cálculos se almacenen en una ubicación de memoria de la computadora. En el ejemplo anterior, a **x** se le asigna un valor de 8. Si usted ingresa el nombre de variable

```
x
```

en MATLAB, obtiene el siguiente resultado:

```
x =
       8
```

Idea clave: el operador asignación es diferente de una igualdad.

El operador asignación es significativamente diferente de una igualdad. Considere el enunciado

```
x = x + 1
```

Éste no es un enunciado algebraico válido, pues claramente **x** no es igual a **x + 1**. Sin embargo, cuando se interpreta como un enunciado de asignación, dice que se debe sustituir el valor actual de **x** almacenado en memoria con un nuevo valor que es igual a la antigua **x** más **1**.

Dado que el valor almacenado en **x** originalmente fue 8, el enunciado regresa

```
x =
       9
```

lo que indica que el valor almacenado en la ubicación de memoria llamada **x** cambió a 9. El enunciado de asignación es similar al proceso familiar de guardar un archivo. Cuando guarda por primera vez un documento en un procesador de palabras, usted le asigna un nombre. Subsecuentemente, después de que realiza cambios, vuelve a guardar su archivo, pero todavía le asigna el mismo nombre. Las versiones primera y segunda no son iguales: sólo asignó una nueva versión de su documento a una ubicación de memoria existente.

Orden de las operaciones

En todos los cálculos matemáticos es importante entender el orden en el que se realizan las operaciones. MATLAB sigue las reglas algebraicas estándar para el orden de operación:

- Primero realiza los cálculos adentro de paréntesis, desde el conjunto más interno hasta el más externo.
- A continuación, realiza operaciones de exponenciación.
- Luego realiza operaciones de multiplicación y división de izquierda a derecha.
- Finalmente, realiza operaciones de suma y resta de izquierda a derecha.

Para entender mejor la importancia del orden de las operaciones, considere los cálculos que se involucran al encontrar el área de un cilindro circular recto.

El área es la suma de las áreas de las dos bases circulares y el área de la superficie curva entre ellas, como se muestra en la figura 2.5. Si la altura (height) del cilindro es 10 cm y el radio (radius) es de 5 cm, se puede usar el siguiente código MATLAB para encontrar el área (surface_area):

```
radius = 5;
height = 10;
surface_area = 2*pi*radius^2 + 2*pi*radius*height
```

El código regresa

```
surface_area =
             471.2389
```

En este caso, MATLAB realiza primero la exponenciación y eleva el radio a la segunda potencia. Luego trabaja de izquierda a derecha y calcula el primer producto y luego el

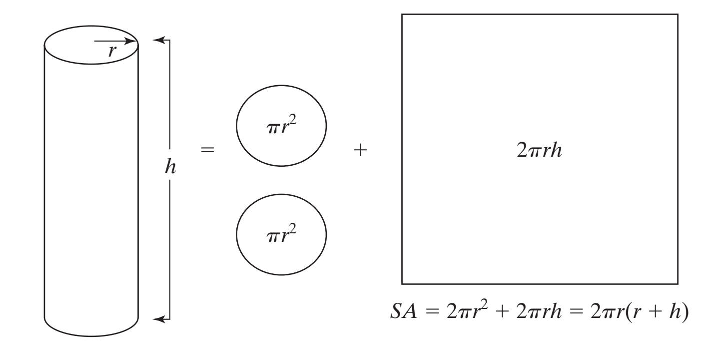


Figura 2.5
Encontrar el área de un cilindro circular recto involucra suma, multiplicación y exponenciación.

segundo producto. Finalmente, suma los dos productos. En vez de ello, podría formular la expresión como

```
surface_area = 2*pi*radius*(radius + height)
```

que también regresa

```
surface_area =
      471.2389
```

En este caso, MATLAB primero encuentra la suma del radio y la altura y luego realiza las multiplicaciones, trabajando de izquierda a derecha. Si olvida incluir los paréntesis, tendría

```
surface_area = 2*pi*radius*radius + height
```

en cuyo caso el programa primero tendría que calcular el producto de **2*pi*radius*radius** y luego sumar **height**, lo que obviamente resulta en la respuesta incorrecta. Note que fue necesario incluir el operador multiplicación antes de los paréntesis, porque MATLAB no supone operador alguno e interpretaría la expresión

```
radius(radius + height)
```

como la función **radius** con entrada (**radius + height**). Puesto que no hay función radio en MATLAB, esta interpretación resultaría en un enunciado de error.

Es importante tener mucho cuidado al convertir ecuaciones en enunciados MATLAB. No hay penalización por agregar paréntesis adicionales y con frecuencia esto hace al código más fácil de interpretar, tanto para el programador como para otros que puedan usar el código en el futuro.

Otra forma de hacer más legible un código de cómputo es descomponer las expresiones largas en múltiples enunciados. Por ejemplo, considere la ecuación

$$f = \frac{\log(ax^2 + bx + c) - \sin(ax^2 + bx + c)}{4\pi x^2 + \cos(x - 2) * (ax^2 + bx + c)}$$

Sería muy fácil cometer un error de tecla en esta ecuación. Para minimizar la posibilidad de que ello ocurra, descomponga la ecuación en muchas piezas. Por ejemplo, primero asigne valores para **x**, **a**, **b** y **c**:

```
x = 9;
a=1;
b=3;
c=5;
```

Luego defina un polinomio y el denominador:

```
poly = a*x^2 + b*x + c;
denom = 4*pi*x^2 + cos(x - 2)*poly;
```

Combine estos componentes en una ecuación final:

```
f=(log(poly) - sin(poly))/denom
```

El resultado es

```
f =
    0.0044
```

Como se mencionó, este enfoque minimiza su oportunidad de error. En lugar de teclear el polinomio tres veces (y arriesgarse a un error cada vez), sólo necesita teclear una sola vez. Aumenta la probabilidad de crear código MATLAB preciso y es más fácil que otros lo entiendan.

Idea clave: intente minimizar su oportunidad de error.

Sugerencia

MATLAB no lee “espacio blanco”, así que no importa si agrega espacios a sus comandos. Es más fácil leer una expresión larga si agrega un espacio antes y después de los signos más (+) y menos (−), pero no antes y después de los signos de multiplicación (*) y división (/).

Ejercicio de práctica 2.3

Prediga los resultados de las siguientes expresiones MATLAB y luego verifique sus predicciones al teclear las expresiones en la ventana de comandos:

1. 6/6 + 5
2. 2*6^2
3. (3+5)*2
4. 3 + 5*2
5. 4*3/2*8
6. 3−2/4+6^2
7. 2^3^4
8. 2^(3^4)
9. 3^5+2
10. 3^(5+2)

Cree y pruebe la sintaxis MATLAB para evaluar las siguientes expresiones y luego verifique sus respuestas con una calculadora de mano.

11. $\dfrac{5 + 3}{9 - 1}$
12. $2^3 - \dfrac{4}{5 + 3}$
13. $\dfrac{5^{2+1}}{4 - 1}$
14. $4\dfrac{1}{2} * 5\dfrac{2}{3}$
15. $\dfrac{5 + 6 * \dfrac{7}{3} - 2^2}{\dfrac{2}{3} * \dfrac{3}{3*6}}$

EJEMPLO 2.1

Operaciones escalares

Los túneles de viento (véase la figura 2.6) juegan un importante papel en la comprensión del comportamiento de las aeronaves de alto rendimiento. Con la finalidad de interpretar los datos del túnel de viento, los ingenieros requieren entender cómo se comportan los gases. La ecuación básica que describe las propiedades de los gases es la ley del gas ideal, una relación que se estudia con detalle en las clases de química de primer año. La ley establece que

$$PV = nRT$$

donde
P = presión en kPa,
V = volumen en m^3,
n = número de kmoles de gas en la muestra,
R = constante de gas ideal, 8.314 kPa m^3/kmol K, y
T = temperatura, en grados kelvin (K).

Además, se sabe que el número de kmoles de gas es igual a la masa del gas dividida por la masa molar (también conocida como peso molecular) o

$$n = m/\mathrm{MW}$$

donde

m = masa en kg y
MW = masa molar en kg/kmol.

En las ecuaciones se pueden usar diferentes unidades si el valor de R se cambia en concordancia.

Suponga ahora que usted sabe que el volumen de aire en el túnel de viento es de 1000 m^3. Antes de que el túnel de viento se encienda, la temperatura del aire es de 300 K, y la presión es de 100 kPa. La masa molar (peso molecular) promedio del aire es aproximadamente 29 kg/kmol. Encuentre la masa del aire en el túnel de viento.

Para resolver este problema, use la siguiente metodología para resolución de problemas:

1. Establezca el problema.
   Cuando usted resuelve un problema, es buena idea volver a enunciarlo en sus propias palabras: calcular la masa del aire en un túnel de viento.

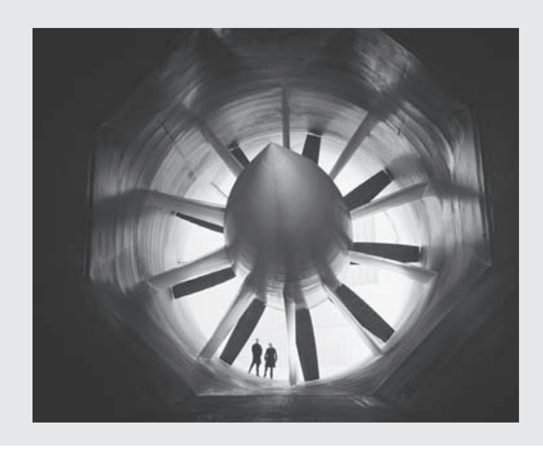


Figura 2.6
Los túneles de viento se usan para probar el diseño de las aeronaves.
(Cortesía de Louis Bencze/Stone/Getty Images Inc.)

2. Describa la entrada y la salida.

   Entrada

Volumen	$V = 1000\ m^3$
Temperatura	$T = 300$ K
Presión	$P = 100$ kPa
Peso molecular	MW = 29 kg/kmol
Constante de gas	$R = 8.314$ kPa m^3/kmol K

   Salida

Masa	m = ¿? Kg

3. Desarrolle un ejemplo a mano.
   Trabajar el problema a mano (o con una calculadora) le permite subrayar un algoritmo, que usted puede traducir más tarde a código MATLAB. Debe elegir datos simples que hagan fácil la comprobación de su trabajo. En este problema, se conocen dos ecuaciones que relacionan los datos:

   $$PV = nRT \quad \text{ley del gas ideal}$$
   $$n = m/\text{MW} \quad \text{conversión de masa a moles}$$

   Resuelva la ley del gas ideal para n y sustituya los valores dados:

   $$\begin{aligned} n &= Pv/RT \\ &= (100\ \text{kPa} \times 1000\ \text{m}^3)/(8.314\ \text{kPa m}^3/\text{kmol K}) \times 300\text{K} \\ &= 40.0930\ \text{kmol} \end{aligned}$$

   Convierta moles a masa al resolver la ecuación de conversión para la masa m y sustituya los valores:

   $$m = n \times \text{MW} = 40.0930\ \text{kmol} \times 29\ \text{kg/kmol}$$
   $$m = 1162.70\ \text{kg}$$

4. Desarrolle una solución MATLAB.
   Primero, limpie la pantalla y la memoria:

   ```
   clear, clc
   ```

   Ahora realice los siguientes cálculos en la ventana de comandos:

   ```
   P = 100
   P =
          100
   T = 300
   T =
          300
   V = 1000
   V =
          1000
   MW = 29
   MW =
          29
   R = 8.314
   R =
          8.3140
   ```

```
n=(P*V)/(R*T)
n =
        40.0930
m = n*MW
m =
        1.1627e+003
```

Existen muchas cosas que usted debe notar acerca de esta solución MATLAB. Primero, puesto que no se utilizaron puntos y comas para suprimir la salida, los valores de las variables se repiten después de cada enunciado de asignación. Note también el uso de paréntesis en el cálculo de *n*. Son necesarios en el denominador, pero no en el numerador. Sin embargo, usar paréntesis en ambos hace al código más fácil de leer.

5. Ponga a prueba la solución.
   En este caso, es suficiente comparar el resultado con el que se obtuvo a mano. Los problemas más complicados resueltos en MATLAB deben usar una diversidad de datos de entrada para confirmar que su solución funciona en una diversidad de casos. En la figura 2.7 se muestra la pantalla MATLAB que se usó para resolver este problema.

   Note que las variables que se definieron en la ventana de comandos se citan en la ventana del área de trabajo. Note también que la historia de comandos lista los comandos que se ejecutaron en la ventana de comandos. Si se desplaza por la ventana de historia de comandos, verá comandos de sesiones previas de MATLAB. Todos estos comandos están disponibles para que usted los mueva a la ventana de comandos.

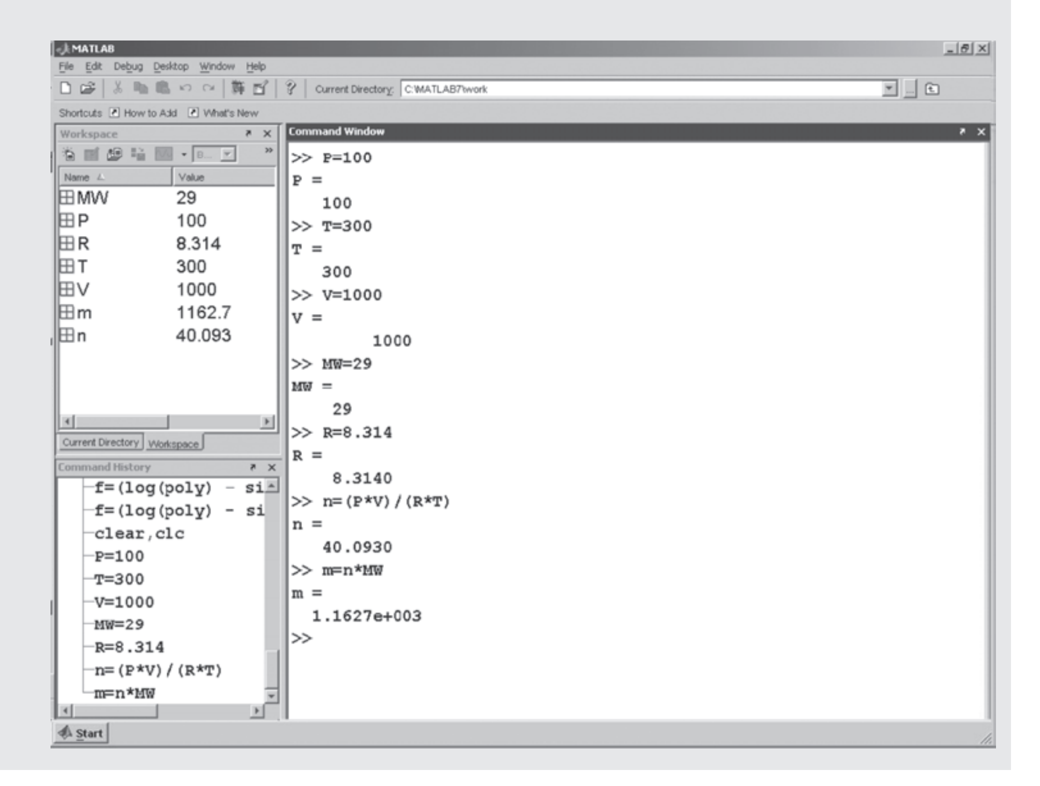


Figura 2.7
Pantalla de MATLAB que se usó para resolver el problema del gas ideal.

Operaciones de arreglos

Usar MATLAB como una calculadora glorificada está bien, pero su verdadera fortaleza está en las manipulaciones matriciales. Como se describió anteriormente, la forma más simple de definir una matriz es usar una lista de números, llamada *lista explícita*. El comando

lista explícita: lista que identifica cada miembro de una matriz

```
x= [1 2 3 4]
```

regresa el vector fila

```
x =
       1 2 3 4
```

Recuerde que, al definir este vector, puede hacer la lista de valores con o sin comas. Una nueva fila se indica mediante un punto y coma, de modo que un vector columna se especifica como

```
y= [ 1; 2; 3; 4]
```

y una matriz que contiene tanto filas como columnas se crearía con el enunciado

```
a = [ 1 2 3 4; 2 3 4 5 ; 3 4 5 6]
```

y regresaría

```
a =
       1 2 3 4
       2 3 4 5
       3 4 5 6
```

Sugerencia

Es más fácil seguir la pista de cuántas variables ingresó en una matriz si ingresa cada fila en una línea separada:

a = [1 2 3 4;

2 3 4 5;

3 4 5 6]

Aunque una matriz complicada tiene que ingresarse a mano, las matrices con intervalos regulares se pueden ingresar mucho más fácilmente. El comando

```
b= 1:5
```

y el comando

```
b = [1:5]
```

regresan ambos una matriz fila

```
b =
     1 2 3 4 5
```

(Los corchetes son opcionales.) El incremento por defecto es 1, pero si usted quiere usar un incremento diferente, colóquelo entre el primero y último valores en el lado derecho del comando. Por ejemplo,

```
c= 1:2:5
```

indica que el incremento entre los valores será 2 y regresa

```
c =
       1      3      5
```

Si usted quiere que MATLAB calcule el espaciamiento entre los elementos, puede usar el comando **linspace**. Especifique el valor inicial, el valor final y cuántos valores quiere en total. Por ejemplo,

```
d=linspace(1,10, 3)
```

regresa un vector con tres valores, espaciados igualmente entre 1 y 10:

```
d =
        1      5.5    10
```

Puede crear vectores espaciados logarítmicamente con el comando **logspace**, que también requiere tres entradas. Los primeros dos valores son potencias de 10 que representan los valores inicial y final en el arreglo. El valor final es el número de elementos en el arreglo. Por ende,

```
e=logspace(1,3,3)
```

regresa tres valores

```
e =
     10   100   1000
```

Note que el primer elemento en el vector es 10^1 y el último elemento en el arreglo es 10^3.

> **Sugerencia**
>
> Puede incluir operaciones matemáticas dentro de un enunciado de definición de matriz. Por ejemplo, puede tener **a = [0: pi/10: pi]**.

Las matrices se pueden usar en muchos cálculos con escalares. Si **a = [1 2 3]** se puede sumar 5 a cada valor en la matriz con la sintaxis

```
b = a + 5
```

que regresa

```
b =
        6      7      8
```

Idea clave: la multiplicación de matrices es diferente de la multiplicación elemento por elemento.

Este enfoque funciona bien para suma y resta; sin embargo, la multiplicación y la división son un poco diferentes. En matemáticas matriciales, el operador de multiplicación (*) tiene un significado específico. Puesto que todas las operaciones MATLAB pueden involucrar matrices, es necesario un operador diferente para indicar multiplicación elemento por elemento. Dicho operador es **.*** (que se llama *multiplicación punto*). Por ejemplo,

```
a.*b
```

resulta en elemento 1 de la matriz **a** multiplicado por el elemento 1 de la matriz **b**,
elemento 2 de la matriz **a** multiplicado por el elemento 2 de la matriz **b**,
elemento n de la matriz **a** multiplicado por el elemento n de la matriz **b**.

Para el caso particular de **a** (que es **[1 2 3]**) y **b** (que es **[6 7 8]**),

```
a.*b
```

regresa

```
ans =
       6      14      24
```

(Haga las operaciones para convencerse de que son las respuestas correctas.)

Sólo usar * implica una multiplicación matricial, que en este caso regresaría un mensaje de error porque **a** y **b** no satisfacen aquí las reglas para multiplicación en álgebra matricial. La moraleja es: tenga cuidado al usar el operador correcto cuando quiera realizar multiplicación elemento por elemento (también llamado arreglo).

La misma sintaxis se cumple para la división elemento por elemento (**./**) y la exponenciación (**.^**) de elementos individuales:

```
a./b
a.^2
```

Como ejercicio, prediga los valores que resultan de las dos expresiones anteriores y luego pruebe sus predicciones mediante la ejecución de los comandos en MATLAB.

Ejercicio de práctica 2.4

Conforme realice los siguientes cálculos, recuerde la diferencia entre los operadores * y **.***, así como los operadores / y **./** y entre ^ y **.^**:

1. Defina la matriz $a = [2.3 \quad 5.8 \quad 9]$ como una variable MATLAB.
2. Encuentre el seno de **a**.
3. Sume 3 a cada elemento en **a**.
4. Defina la matriz $b = [5.2 \quad 3.14 \quad 2]$ como una variable MATLAB.
5. Sume cada elemento de la matriz **a** y la matriz **b**.
6. Multiplique cada elemento en **a** por el correspondiente elemento en **b**.
7. Eleve al cuadrado cada elemento en la matriz **a**.
8. Cree una matriz llamada **c** de valores igualmente espaciados, desde 0 hasta 10, con un incremento de 1.
9. Cree una matriz llamada **d** de valores igualmente espaciados, desde 0 hasta 10, con un incremento de 2.
10. Use la función **linspace** para crear una matriz de seis valores igualmente espaciados, desde 10 hasta 20.
11. Use la función **logspace** para crear una matriz de cinco valores logarítmicamente espaciados entre 10 y 100.

Idea clave: la capacidad matricial de MATLAB hace sencillo realizar cálculos repetitivos.

La capacidad matricial de MATLAB hace fácil realizar cálculos repetitivos. Por ejemplo, suponga que tiene una lista de ángulos en grados que le gustaría convertir a radianes. Primero ponga los valores en una matriz. Para ángulos de 10, 15, 70 y 90, ingrese

```
degrees = [ 10 15 70 90];
```

Para cambiar los valores a radianes, debe multiplicar por $\pi/180$:

```
radians=degrees*pi/180
```

Este comando regresa una matriz llamada **radians**, con los valores en radianes. (¡Inténtelo!) En este caso, podría usar o el operador * o el **.***, porque la multiplicación involucra una sola matriz (**degrees**) y dos escalares (pi y 180). Por lo tanto, podría haber escrito

```
radians=degrees.*pi/180
```

Sugerencia

En MATLAB, el valor de π se construye como un número punto flotante llamado **pi**. Puesto que π es un número irracional, no se puede expresar *exactamente* con una representación en punto flotante, de modo que la constante **pi** de MATLAB en realidad es una aproximación. Puede ver esto cuando encuentra **sin(pi)**. De la trigonometría, la respuesta debería ser 0. Sin embargo, MATLAB regresa un número muy pequeño. El valor real depende de su versión del programa: la versión profesional 7 regresó 1.2246e-016. En la mayoría de los cálculos, esto no haría una diferencia en el resultado final.

Otro operador matricial útil es la transposición. El operador transpuesto cambia filas a columnas y viceversa. Por ejemplo,

```
degrees'
```

regresa

```
ans =
      10
      15
      70
      90
```

Esto facilita la creación de tablas. Por ejemplo, para crear una tabla que convierta grados a radianes, ingrese

```
table =[degrees' , radians']
```

que le pide a MATLAB crear una matriz llamada **table**, en la que la columna 1 es degrees (grados) y la columna 2 es radians (radianes):

```
table =
      10.0000    0.1745
      15.0000    0.2618
      70.0000    1.2217
      90.0000    1.5708
```

Si traspone una matriz bidimensional, todas las filas se convierten en columnas y todas las columnas se convierten en filas. Por ejemplo, el comando

```
table'
```

resulta en

```
10.0000   15.0000   70.0000   90.0000
 0.1745    0.2618    1.2217    1.5708
```

Note que **table** no es un comando MATLAB, sino meramente un nombre de variable conveniente. Se podría haber usado cualquier nombre significativo, por decir, **conversiones** o **grados_a_radianes**.

EJEMPLO 2.2

Cálculos matriciales con escalares

Los datos científicos, como los que se recolectan de los túneles de viento, usualmente están en unidades SI (Système International). Sin embargo, mucha de la infraestructura fabril de Estados Unidos se nominó en unidades inglesas (a veces llamadas Ingeniería Americana o Estándar Americano). Los ingenieros necesitan estar familiarizados con ambos sistemas y deben ser especialmente cuidadosos cuando comparten datos con otros ingenieros. Acaso el ejemplo más notorio de los problemas de confusión de unidades es el Mars Climate Orbiter (figura 2.8), que fue el segundo vuelo del programa Mars Surveyor. La nave espacial se quemó en la órbita de Marte, en septiembre de 1999, debido a una tabla de referencia incrustada en el software de la nave. La tabla, probablemente generada a partir de las pruebas en el túnel de viento, usaba libras fuerza (lbf) cuando el programa esperaba valores en newtons (N).

En este ejemplo, se usará MATLAB para crear una tabla de conversión de libras fuerza a newtons. La tabla comenzará en 0 e irá hasta 1000 lbf, en intervalos de 100 lbf. El factor de conversión es

$$1 \text{ lbf} = 4.4482216 \text{ N}$$

1. Establezca el problema.
   Cree una tabla que convierta libras fuerza (lbf) a newtons (N).
2. Describa las entradas y salidas.

   Entrada

Valor inicial en la tabla	0 lbf
Valor final en la tabla	1000 lbf
Incremento entre valores	100 lbf
La conversión de lbf a N es	1 lbf = 4.4482216 N

   Salida

   Tabla que presenta libras fuerza (lbf) y newtons (N)
3. Desarrolle un ejemplo a mano.
   Puesto que se creará una tabla, tiene sentido verificar algunos valores diferentes. Al elegir números para los que la matemática sea sencilla hace que el ejemplo a mano sea simple para completar, pero todavía valioso como comprobación:

$$\begin{aligned} 0 &* 4.4482216 = 0 \\ 100 &* 4.4482216 = 444.82216 \\ 1000 &* 4.4482216 = 4448.2216 \end{aligned}$$

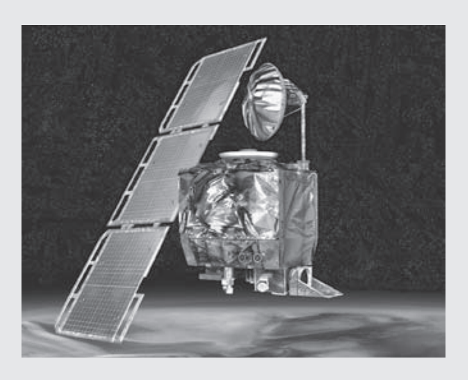


Figura 2.8
Mars Climate Orbiter. (Cortesía de NASA/Jet Propulsion Laboratory.)

4. Desarrolle una solución en MATLAB.

```
clear, clc
lbf = [0:100:1000];
N = lbf * 4.44822;
[lbf',N']
ans =
  1.0e+003 *
       0       0
    0.1000    0.4448
    0.2000    0.8896
    0.3000    1.3345
    0.4000    1.7793
    0.5000    2.2241
    0.6000    2.6689
    0.7000    3.1138
    0.8000    3.5586
    0.9000    4.0034
    1.0000    4.4482
```

Siempre es una buena idea limpiar tanto el área de trabajo como la ventana de comandos antes de comenzar un nuevo problema. Note en la ventana del área de trabajo (figura 2.9) que **lbf** y **N** son matrices de 1 × 11 y que **ans** (que es donde se almacena la tabla creada) es una matriz de 11 × 2. La salida de los primeros dos comandos se suprimió al agregar un punto y coma al final de cada línea. Sería muy fácil crear una tabla con más entradas al cambiar el incremento a 10 o incluso a 1. Note también que necesitará multiplicar los resultados que se muestran en la tabla por 1000 para obtener las respuestas correctas. MATLAB le dice que esto es necesario directamente arriba de la tabla, donde se muestra el factor de escala común.

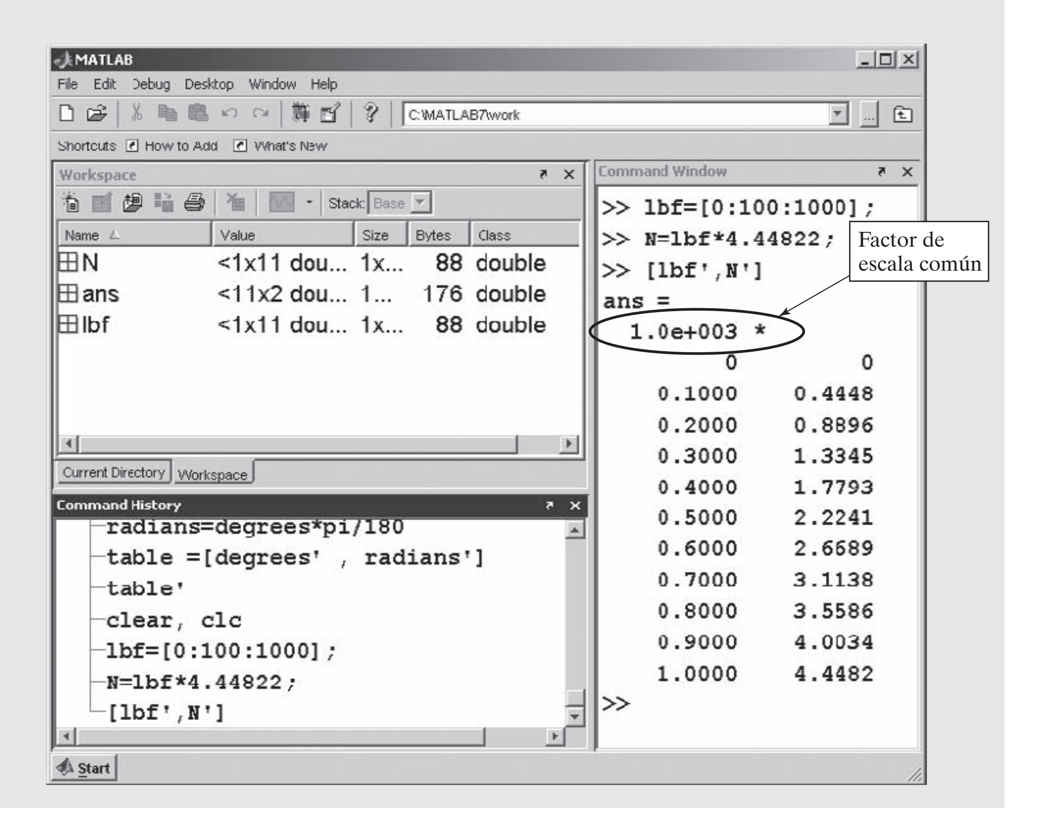


Figura 2.9
La ventana del área de trabajo MATLAB muestra las variables conforme se crean.

5. Ponga a prueba la solución.
   La comparación de los resultados de la solución MATLAB con la solución a mano muestra que son iguales. Una vez verificado que la solución funciona, es fácil usar el mismo algoritmo para crear otras tablas de conversión. Por ejemplo, modifique este ejemplo para crear una tabla que convierta newtons (N) a libras fuerza (lbf), con un incremento de 10 N, desde 0 N hasta 1000 N.

EJEMPLO 2.3

Cálculo del arrastre

Una característica de rendimiento que se puede determinar en un túnel de viento es el arrastre (drag). La fricción relacionada con el arrastre en el Mars Climate Observer (producida por la atmósfera de Marte) resultó en el incendio de la nave espacial durante las correcciones de curso. El arrastre también es extremadamente importante en el diseño de aeronaves terrestres. (Véase la figura 2.10.)

El arrastre es la fuerza que se genera conforme un objeto, como un avión, se mueven a través de un fluido. Desde luego, en el caso de un túnel de viento, el aire pasa sobre un modelo estacionario, pero las ecuaciones son las mismas. El arrastre es una fuerza complicada que depende de muchos factores. Un factor es la fricción de piel, que es una función de las propiedades de la superficie de la aeronave, las propiedades del fluido en movimiento (aire en este caso) y de los patrones de flujo provocados por la forma de la aeronave (o, en el caso del Mars Climate Observer, por la nave espacial). El arrastre se puede calcular con la ecuación de arrastre

$$\text{arrastre} = C_d \frac{\rho V^2 A}{2}$$

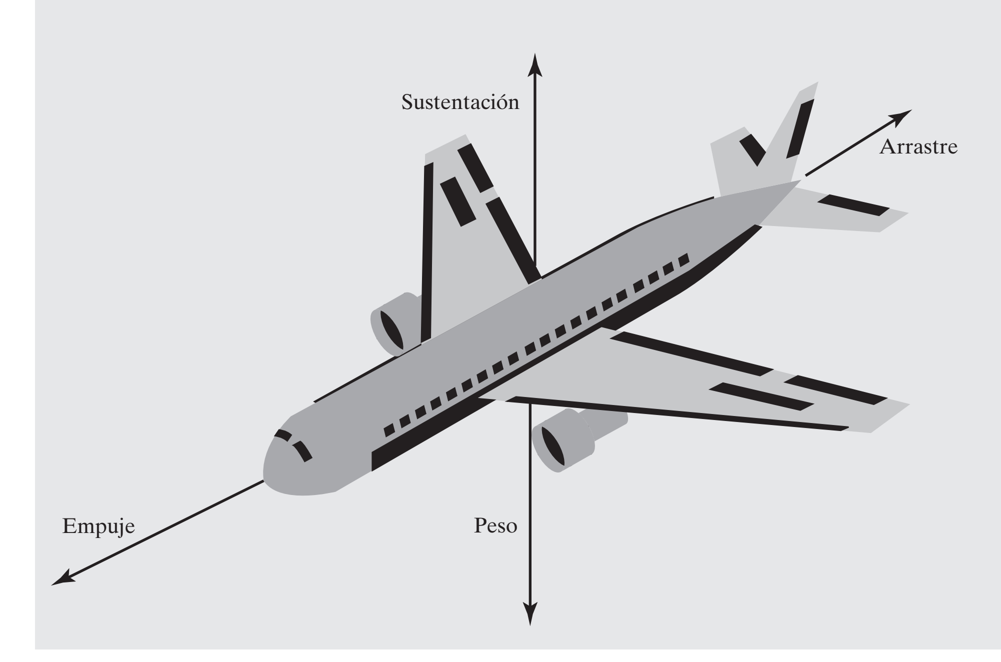


Figura 2.10
El arrastre es una fuerza mecánica generada por un objeto sólido que se mueve a través de un fluido.

donde C_d = coeficiente de arrastre (drag), que se determina experimentalmente, por lo general en un túnel de viento,
ρ = densidad del aire,
V = velocidad de la aeronave,
A = área de referencia (el área superficial sobre la que fluye el aire).

Aunque el coeficiente de arrastre no es una constante, se puede considerar como constante a rapideces bajas (menores a 200 mph). Suponga que los siguientes datos se midieron en un túnel de viento:

arrastre (drag)	20,000 N
ρ	1×10^{-6} kg/m^3
V	100 mph (necesitará convertir esto a metros por segundo)
A	1 m^2

Calcule el coeficiente de arrastre. Finalmente, use este coeficiente de arrastre determinado experimentalmente para predecir cuánto arrastre se ejercerá sobre la aeronave a rapideces desde 0 mph hasta 200 mph.

1. Establezca el problema.
   Calcule el coeficiente de arrastre sobre la base de los datos recopilados en un túnel de viento. Use el coeficiente de arrastre para determinar el arrastre a varias rapideces.
2. Describa las entradas y salidas.

   Entrada

Arrastre (drag)	20,000 N
Densidad del aire ρ	1×10^{-6} kg/m^3
Rapidez V	100 mph
Área de la superficie A	1 m^2

   Salida

   Coeficiente de arrastre
   Arrastre a rapideces de 0 a 200 mph
3. Desarrolle un ejemplo a mano.
   Primero encuentre el coeficiente de arrastre a partir de los datos experimentales. Note que la rapidez está en millas/h y se debe cambiar a unidades consistentes con el resto de los datos (m/s). ¡Jamás se enfatizará demasiado la importancia de acarrear unidades en los cálculos de ingeniería!

$$C_d = \text{arrastre} \times 2/(\rho \times V^2 \times A)$$

$$= \frac{(20{,}000\text{ N} \times 2)}{1 \times 10^{-6}\text{ kg/m}^3 \times \left(100\text{ millas/h} \times 0.4470\,\dfrac{\text{m/s}}{\text{millas/h}}\right)^2 \times 1\text{ m}^2}$$

$$= 2.0019 \times 10^7$$

Dado que un newton es igual a un kg m/s^2, el coeficiente de arrastre es adimensional. Ahora use el coeficiente de arrastre para encontrar el arrastre a diferentes rapideces:

$$\text{arrastre} = C_d \times \rho \times V^2 \times A/2$$

Con una calculadora, encuentre el valor del arrastre con $V = 200$ mph:

$$\text{arrastre} = \frac{2.0019 \times 10^7 \times 1 \times 10^{-6}\ \text{kg/m}^3 \times \left(200\ \text{millas/h} \times 0.4470 \dfrac{\text{m/s}}{\text{millas/h}}\right)^2 \times 1\ \ \text{m}^2}{2}$$

arrastre = 80,000 N

4. Desarrollo una solución MATLAB.

```
drag = 20000;            Define las variables y cambia V a unidades SI.
r = 0.000001;            Aquí se usó r en lugar de ρ para la densidad.
V = 100*0.4470;
A = 1;
cd = drag*2/(r*V^2*A)    Calcula el coeficiente de arrastre.
cd =
  2.0019e+007
V = 0:20:200;            Redefine V como matriz.
V = V*0.4470;            La cambia a unidades SI y calcula
drag = cd*r*V.^2*A/2;    el arrastre.
table = [V', drag']
table =
  1.0e+004 *
       0         0
    0.0009    0.0800
    0.0018    0.3200
    0.0027    0.7200
    0.0036    1.2800
    0.0045    2.0000
    0.0054    2.8800
    0.0063    3.9200
    0.0072    5.1200
    0.0080    6.4800
    0.0089    8.0000
```

Note que la ecuación para el arrastre, o

```
drag = cd * r * V.^2 * A/2;
```

usa el operador **.^**, porque se pretende que cada valor en la matriz **V** sea al cuadrado, no que toda la matriz **V** se multiplique por sí misma. Usar sólo el operador exponenciación (^) resultaría en un mensaje de error. Desafortunadamente, es posible componer problemas en los que el uso del operador erróneo no proporciona un mensaje de error, sino que da una respuesta equivocada. Esto hace especialmente importante el paso 5 de la metodología para resolver problemas.

5. Ponga a prueba la solución.
   Al comparar la solución a mano con la solución MATLAB (figura 2.11), se ve que ambas dan el mismo resultado. Una vez que se confirma que el algoritmo funciona con los datos de muestra, se pueden sustituir nuevos datos y tener la seguridad de que los resultados serán correctos. De manera ideal, los resultados también se deben comparar con los datos experimentales, para confirmar que las ecuaciones que se utilizan de manera precisa modelan el proceso físico real.

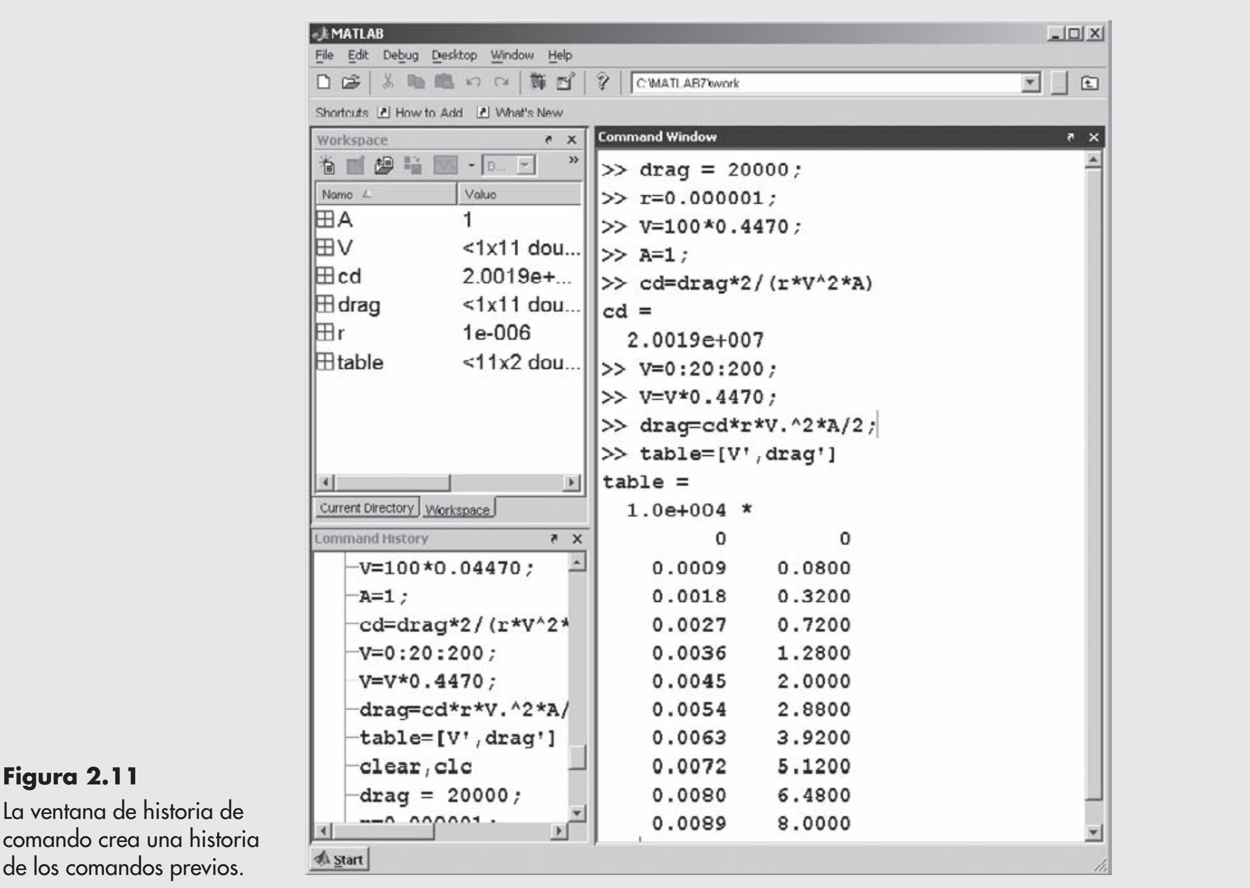


Figura 2.11
La ventana de historia de comando crea una historia de los comandos previos.

2.3.3 Despliegue de números

notación científica: número que se representa como un valor entre uno y diez por diez a una potencia apropiada

Notación científica

Aunque es posible ingresar cualquier número en notación decimal, no siempre es la mejor forma de representar números o muy grandes o muy pequeños. Por ejemplo, un número que se usa frecuentemente en química es la constante de Avogadro, cuyo valor, a cuatro cifras significativas, es 602,200,000,000,000,000,000,000. De igual modo, el diámetro de un átomo de hierro es aproximadamente 140 picómetros, que es .000000000140 metros. La notación científica expresa un valor como un número entre 1 y 10, multiplicado por una potencia de 10 (el exponente). En notación científica, el número de Avogadro se convierte en 6.022×10^{23}, y el diámetro de un átomo de hierro se vuelve 1.4×10^{-10} metros. En MATLAB, los valores en notación científica se designan con una **e** entre el número decimal y el exponente. (Probablemente su calculadora usa notación similar.) Por ejemplo, usted puede tener

```
Avogadros_constant = 6.022e23  ;
Iron_diameter = 140e-12;  o
Iron_diameter = 1.4e-10;
```

Es importante omitir los espacios en blanco entre el número decimal y el exponente. Por ejemplo, MATLAB interpretará

```
6.022   e23
```

como dos valores (6.022 y 10^{23}).

Sugerencia

Aunque es una convención común usar **e** para identificar una potencia de 10, los estudiantes (y el profesor) a veces confunden esta nomenclatura con la constante matemática *e*, que es igual a 2.7183. Para elevar *e* a una potencia, use la función **exp**.

Formato de despliegue

En MATLAB están disponibles algunos formatos de despliegue. No importa cuál formato de despliegue elija, MATLAB usa en sus cálculos números punto flotante de doble precisión. Exactamente cuántos dígitos se usan depende de su cálculo. Sin embargo, cambiar el formato de despliegue no cambia la precisión de sus resultados. A diferencia de algunos otros programas, MATLAB maneja los números enteros y decimales como números de punto flotante.

Idea clave: MATLAB no diferencia entre números enteros y de punto flotante, a menos que se invoquen funciones especiales.

Cuando los elementos de una matriz se despliegan en MATLAB, los enteros siempre se imprimen sin punto decimal. No obstante, los valores con fracciones decimales se imprimen en el formato corto por defecto que muestra cuatro dígitos decimales. Por ende,

```
A = 5
```

regresa

```
A =
      5
```

pero

```
A = 5.1
```

Idea clave: no importa cuál formato de despliegue se seleccione, los cálculos se realizan usando números de punto flotante con doble precisión.

regresa

```
A =
      5.1000
```

y

```
A = 51.1
```

regresa

```
A =
      51.1000
```

MATLAB le permite especificar otros formatos que muestren dígitos significativos adicionales. Por ejemplo, para especificar que usted quiere que los valores se desplieguen en un formato decimal con 14 dígitos decimales, use el comando

```
format long
```

que cambia todos los despliegues subsecuentes. Por ende, con especificación **format long**,

```
A
```

ahora regresa

```
A =
      51.10000000000000
```

Cuando el formato se especifica como **format bank** (formato banco), se despliegan dos dígitos decimales:

```
A =
     51.10
```

Usted puede regresar el formato a cuatro dígitos decimales con el comando

```
format short
```

Para verificar los resultados, usted puede recordar el valor de **A**:

```
A
A =
       51.1000
```

Cuando los números se vuelven demasiado largos o demasiado pequeños para que MATLAB los despliegue en el formato por defecto, automáticamente los expresa en notación científica. Por ejemplo, si usted ingresa la constante de Avogadro en MATLAB en notación decimal como

```
a=602000000000000000000000
```

el programa regresa

```
a =
       6.0200e+023
```

Usted puede forzar a MATLAB a desplegar *todos* los números en notación científica con **format short e** (con cuatro dígitos decimales) o **format long e** (con 14 dígitos decimales. Por ejemplo,

```
format short e
x = 10.356789
```

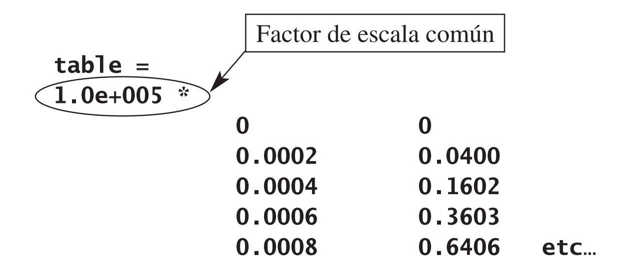
regresa

```
x =
       1.0357e+001
```

Con los formatos largo y corto se aplica un factor de escala común a toda la matriz, si los elementos se vuelven muy largos o muy cortos. Este factor de escala se imprime junto con los valores escalados. Por ejemplo, cuando la ventana de comando regresa a

```
format short
```

los resultados del ejemplo 2.3 se despliegan como



Otro par de formatos que ocasionalmente pueden ser útiles son **format +** y **format rat**. Cuando una matriz se despliega en **format +**, los únicos caracteres que se imprimen son los

Tabla 2.2 Formatos de despliegue numérico

Comando MATLAB	Despliegue	Ejemplo
`format short`	4 dígitos decimales	`3.1416`
`format long`	14 dígitos decimales	`3.14159265358979`
`format short e`	4 dígitos decimales	`3.1416e+000`
`format long e`	14 dígitos decimales	`3.141592653589793e+000`
`format bank`	2 dígitos decimales	`3.14`
`format +`	+, −, espacio en blanco	+
`format rat`	forma fraccional	`355/113`

signos más y menos. Si un valor es positivo, se desplegará un signo más; si un valor es negativo, se desplegará un signo menos. Si un valor es cero, no se desplegará nada. Este formato permite visualizar una matriz grande en términos de sus signos:

```
format +
B = [1, -5, 0, 12; 10005, 24, -10,4]
B =
        +- +
        ++-+
```

El comando **format rat** despliega números como números racionales (es decir: como fracciones). En consecuencia,

número racional: número que se puede representar como una fracción

```
format rat
x = 0:0.1:0.5
```

regresa

```
x =
       0   1/10   1/5   3/10   2/5   1/2
```

El comando **format** también le permite controlar cuán apretadamente se espacia la información en la ventana de comando. Por defecto (**format loose**) se inserta un salto de línea entre las expresiones proporcionadas por el usuario y los resultados que regresa la computadora. El comando **format compact** remueve dichos saltos de línea. Los ejemplos en este texto usan el formato compacto para ahorrar espacio. La tabla 2.2 muestra cómo se despliega el valor de π en cada formato.

Si ninguno de estos formatos de despliegue numérico predefinidos son adecuados para usted, puede controlar líneas individuales de salida con la función **fprintf**.

2.4 CÓMO GUARDAR EL TRABAJO

Trabajar en la ventana de comandos es similar a realizar cálculos en su calculadora científica. Cuando apaga la calculadora o cuando sale del programa, su trabajo desaparece. *Es* posible guardar los *valores* de las variables que definió en la ventana de comando y que se enlisten en la ventana del área de trabajo, pero aunque hacerlo es útil, es más probable que usted quiera guardar la lista de comandos que generaron sus resultados. En esta sección, primero se mostrará cómo guardar y recuperar variables (los resultados de las asignaciones que hizo y los cálculos que realizó) a archivos MAT o a archivos DAT. Luego se introducirán los archivos-m script, que se crean en la ventana de edición. Los archivos-m script le permiten guardar una

lista de comandos y ejecutarlos más tarde. Descubrirá que los archivos-m script son especialmente útiles para resolver problemas de tarea en casa.

2.4.1 Cómo guardar variables

Para preservar las variables que creó en la **ventana de comandos** (verifique la lista de variables en la **ventana de área de trabajo** en el lado izquierdo de la pantalla de MATLAB) entre sesiones, debe guardar los contenidos de la **ventana de área de trabajo** a un archivo. El formato por defecto es un archivo binario llamado archivo MAT. Para guardar el área de trabajo (recuerde, éstas son sólo las variables, no la lista de comandos en la ventana de comandos) a un archivo, escriba

```
save < file_name >
```

en el prompt. Aunque **save** es un comando MATLAB, **file_name** es un nombre de archivo definido por el usuario. En este texto, los nombres definidos por el usuario se indicarán colocándolos dentro de paréntesis angulares (< >). El nombre del archivo puede ser cualquier nombre que elija, en tanto esté en conformidad con las convenciones de nomenclatura para las variables en MATLAB. De hecho, incluso ni siquiera necesita proporcionar un nombre de archivo. Si no lo hace, MATLAB nombra al archivo **matlab.mat**. También podría elegir

```
File → Save Workspace As
```

de la barra de menú, que luego lo conminará a ingresar un nombre de archivo para sus datos. Para restaurar un área de trabajo, escriba

```
load < file_name >
```

De nuevo, **load** es un comando MATLAB, pero **file_name** es el nombre de archivo definido por el usuario. Si sólo escribe **load**, MATLAB buscará el archivo **matlab.mat** por defecto.

El archivo que guarde se almacenará en el directorio actual.

Por ejemplo, escriba

```
clear, clc
```

Este comando limpiará tanto el área de trabajo como al ventana de comandos. Verifique que el área de trabajo esté vacía mediante la comprobación de la ventana del área de trabajo o al escribir

```
whos
```

Ahora defina varias variables, por ejemplo

```
a = 5;
b = [1,2,3];
c = [ 1, 2; 3,4];
```

Compruebe de nuevo la ventana del área de trabajo, para confirmar que las variables se almacenaron. Ahora, guarde el área de trabajo a un archivo llamado my_example_file:

```
save my_example_file
```

Confirme que en el directorio actual se almacenó un nuevo archivo. Si prefiere guardar el archivo a otro directorio (por ejemplo, a un disco extraíble), use el botón de navegador (véase la figura 2.2) para navegar hacia el directorio de su elección. Recuerde que, en una computadora pública, probablemente el directorio actual se purgue después de que cada usuario sale del sistema.

Ahora, limpie el área de trabajo y la ventana de comandos escribiendo

```
clear, clc
```

La ventana de área de trabajo debe estar vacía. Puede recuperar las variables perdidas y sus valores al cargar el archivo (my_example_file.mat) de nuevo en el área de trabajo:

```
load my_example_file
```

Recuerde: el archivo que quiera cargar debe estar en el directorio actual, o de otro modo MATLAB no podrá encontrarlo. Escriba en la ventana de comandos

```
a
```

que regresa

```
a =
       5
```

De igual modo,

```
b
```

regresa

```
b =
       1   2   3
```

y al escribir

```
c
```

regresa

```
c =
       1   2
       3   4
```

MATLAB también puede almacenar matrices individuales o listas de matrices en el directorio actual con el comando

```
save <file_name>  <variable_list>
```

donde **file_name** es el nombre de archivo definido por el usuario que designa la ubicación en memoria en la que desea almacenar la información, y donde **variable_list** es la lista de variables a almacenar en el archivo. Por ejemplo,

```
save  my_new_file  a b
```

guardaría sólo las variables **a** y **b** en **my_new_file-mat**.

Si los datos que guardó los usará un programa distinto a MATLAB (como C o C++), el formato .mat no es apropiado porque los archivos .mat son exclusivos de MATLAB. El formato ASCII es estándar entre plataformas de computadoras y es más apropiado si necesita compartir archivos. MATLAB le permite guardar archivos como archivos ASCII al modificar el comando save a

```
save <file_name>  <variable_list>  -ascii
```

El comando **–ascii** le dice a MATLAB que almacene los datos en un formato de texto estándar de ocho dígitos. Los archivos ASCII se guardarán en un archivo .dat en lugar de en un archivo .mat; sólo asegúrese de agregar .dat a su nombre de archivo:

ascii: formato de almacenamiento de datos binarios

```
save my_new_file.dat a b -ascii
```

Si no agrega .dat, MATLAB le pondrá por defecto .mat.

Idea clave: cuando guarda el área de trabajo, usted sólo guarda las variables y sus valores; no guarda los comandos que ejecutó.

Si necesita más precisión, los datos se pueden almacenar en un formato de texto de 16 dígitos:

```
save file_name  variable_list  -ascii  -double
```

También es posible delimitar los elementos (números) con tabuladores:

```
save <file_name>  <variable_list>  -ascii  -double  -tabs
```

Puede recuperar los datos del directorio actual con el comando load:

```
load <file_name>
```

Por ejemplo, para crear la matriz **z** y guardarla al archivo **data_2.dat** en formato de texto de ocho dígitos, use los comandos siguientes:

```
z = [5 3 5; 6 2 3];
save data_2.dat z -ascii
```

En conjunto, dichos comandos hacen que cada fila de la matriz **z** se escriba a una línea separada en el archivo de datos. Usted puede ver el archivo data_2.dat al hacer doble clic en el nombre del archivo en la ventana de directorio actual. (Véase la figura 2.12.) Acaso la forma más sencilla de recuperar datos de un archivo ASCII .dat sea ingresar el comando **load** seguido por el nombre del archivo. Esto hace que la información se lea en una matriz con el mismo nombre que el archivo de datos. Sin embargo, también es muy fácil usar el Import Wizard (asistente de importación) interactivo de MATLAB para cargar los datos. Cuando hace doble clic en el nombre del archivo de datos en el directorio actual para ver el contenido del archivo, el Import Wizard se lanzará automáticamente. Sólo siga las instrucciones para cargar los datos en el área de trabajo con el mismo nombre que el archivo de datos. Puede usar esta misma técnica para importar datos de otros programas, incluso hojas de cálculo Excel, o puede seleccionar **File → Import data...** de la barra de menú.

2.4.2 Archivos-m script

Además de proporcionar un ambiente computacional interactivo (al usar la ventana de comandos como una memoria de trabajo auxiliar), MATLAB contiene un poderoso lenguaje de programación. Como programador, puede crear y guardar código en archivos llamados archivos-m. Un archivo-m es un archivo de texto ASCII similar a los archivos de código fuente de C o FORTRAN. Se puede crear y editar con el editor/debugger (depurador) de archivo-m de MATLAB (la ventana de edición que se analizó en la sección 2.2.7), o puede usar otro editor de texto de su elección. Para abrir la ventana de edición, seleccione

```
File → New → M-file
```

de la barra de menú de MATLAB. En la figura 2.13 se muestra la ventana de edición de MATLAB.

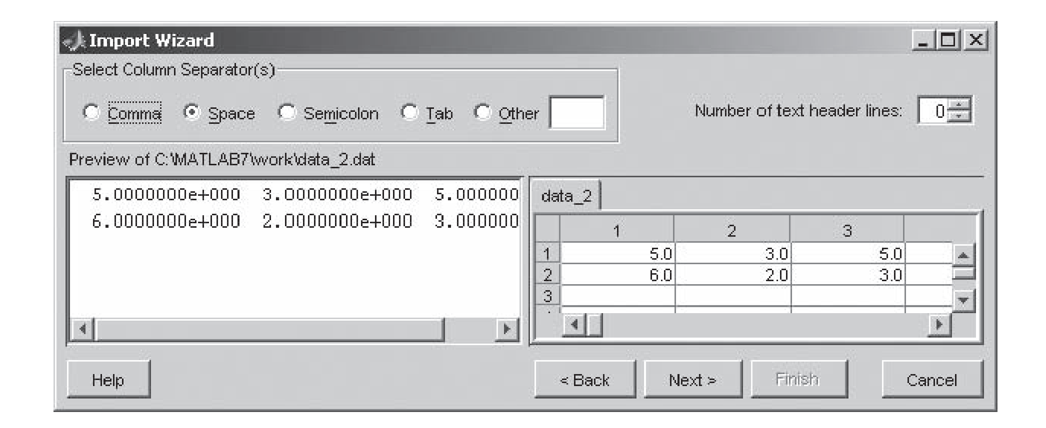


Figura 2.12
Hacer doble clic en el nombre de archivo en el directorio de comandos lanza el Import Wizard.

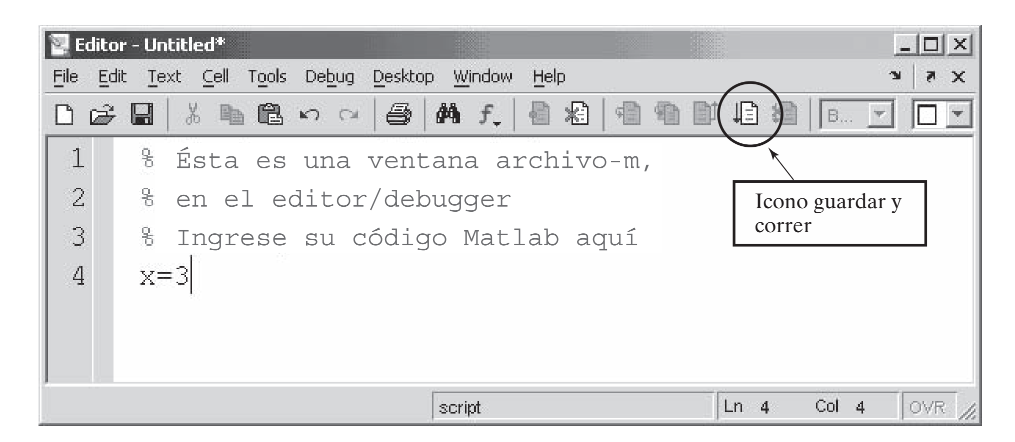


Figura 2.13
Ventana de edición de MATLAB, también llamado editor/debugger.

Si usted elige un editor de texto diferente, asegúrese de que los archivos que guarde sean archivos ASCII. Notepad es un ejemplo de editor de texto que por defecto origina una estructura de archivo ASCII. Otros procesadores de palabra, como WordPerfect o Word, requerirán que usted especifique la estructura ASCII cuando guarde el archivo. Esos programas dan por defecto estructuras de archivo propietario que no se someten a ASCII, y pueden producir algunos resultados inesperados si intenta usar código escrito en ellos si no especifica que los archivos se guarden en formato ASCII.

archivo-m: lista de comandos MATLAB almacenados en un archivo separado

Cuando se guarda un archivo-m, éste se almacena en el directorio actual. Será necesario que nombre su archivo con un nombre variable MATLAB válido, esto es, un nombre que comience con una letra y contenga sólo letras, números y el guión bajo (_). No se permiten los espacios. (Véase la sección 2.3.1.)

Idea clave: los dos tipos de archivos-m son scripts y funciones.

Existen dos tipos de archivos-m, llamados scripts y funciones. Un archivo-m script es simplemente una lista de enunciados MATLAB que se guardan en un archivo (por lo general, con una extensión de archivo .m). El script puede usar cualesquiera variables que se hayan definido en el área de trabajo, y cualesquiera variables que se creen en el script se agregarán al área de trabajo cuando el script termine. Puede ejecutar un script creado en la ventana de edición MATLAB al seleccionar el icono Save and Run (guardar y correr) de la barra de menú, como se muestra en la figura 2.13. De manera alternativa, puede ejecutar un script al escribir un nombre de archivo o al usar el comando run de la ventana de comandos.

Suponga que creó un archivo script llamado myscript.m. Puede correr el script desde la ventana de edición o usar una de tres formas de ejecutar el script desde la ventana de comandos. (Véase la tabla 2.3.) Las tres técnicas son equivalentes. Usted elige cuál usar.

Puede descubrir cuáles son los archivos-m y MAT en el directorio actual al escribir

```
what
```

en la ventana de comandos. También puede navegar en el directorio actual al buscar en la ventana de directorio actual.

Tabla 2.3 Enfoques para ejecutar un archivo-m script desde la ventana de comandos

Comando MATLAB	Comentarios
`myscript`	Escriba el nombre de archivo. Se supone la extensión de archivo .m
`run myscript`	Use el comando run con el nombre de archivo.
`run('myscript')`	Use la forma funcional del comando run

Usar archivos-m script le permite trabajar en un proyecto y guardar la lista de comandos para uso futuro. Puesto que más adelante usará estos archivos, es buena idea salpicarlos libremente con comentarios. El operador comentario en MATLAB es el signo de porcentaje, como en

```
%  Esto es un comentario
```

MATLAB no ejecutará código alguno en una línea comentada.

También puede agregar comentarios después de un comando, pero en la misma línea:

```
a = 5        %La variable a se define como 5
```

El código MATLAB que se podría ingresar en un archivo-m y utilizarse para resolver el ejemplo 2.3 es como sigue:

```
clear, clc
% Archivo-m script para encontrar drag (arrastre)
% Primero defina las variables
drag = 20000;              %Defina drag en newtons
r = 0.000001;              %Defina densidad del aire en kg/m^3
V = 100*0.4470;            %Defina rapidez en m/s
A = 1;                     %Defina área en m^2
% Calcula coeficiente de arrastre
cd = drag *2/(r*V^2*A)
% Encuentra el arrastre para varias rapideces
V = 0:20:200;              %Redefine rapidez
V = V*.4470                %Cambia rapidez a m/s
drag = cd*r*V.^2*A/2;      %Calcula arrastre
table = [V',drag']         %Crea una tabla de resultados
```

Este código se podría correr o desde el archivo-m o desde la ventana de comandos. En cualquier caso, los resultados aparecerán en la ventana de comandos, y las variables se almacenarán en el área de trabajo.

Sugerencia

Puede ejecutar una porción de un archivo-m al resaltar una sección y luego hacer clic derecho y seleccionar **Evaluate Section** (evaluar sección). También desde este menú puede comentar o "descomentar" secciones completas de código; hacerlo así es útil cuando crea programas mientras todavía depura su trabajo.

El ejemplo final de este capítulo usa un archivo-m script para encontrar la rapidez y aceleración que puede alcanzar una nave espacial al salir del sistema solar.

EJEMPLO 2.4

Creación de un archivo-m para calcular la aceleración de una nave espacial

En ausencia de arrastre (drag), los requisitos de potencia de propulsión para una nave espacial se determinan de manera bastante simple. Recuerde de la física básica que

$$F = ma$$

En otras palabras, fuerza (F) es igual a masa (m) por aceleración (a). El trabajo (W) es fuerza por distancia (d), y dado que la potencia (P) es trabajo por unidad de tiempo, la potencia se convierte en fuerza por velocidad (v):

$$W = Fd$$

$$P = \frac{W}{t} = F \times \frac{d}{t} = F \times v = m \times a \times v$$

Esto significa que los requerimientos de potencia para la nave espacial dependen de su masa, de qué tan rápido marche y de qué tan rápidamente necesite acelerar o frenar. Si no se aplica potencia, la nave espacial simplemente sigue viajando a su velocidad actual. En tanto no se quiera hacer algo rápidamente, las correcciones de curso se pueden realizar con muy poca potencia. Desde luego, la mayoría de los requerimientos de potencia para la nave espacial no se relacionan con la navegación. La potencia se requiere para comunicación, para mantenimiento y para experimentos y observaciones científicos.

Las naves espaciales *Voyager 1* y *2* exploraron el sistema solar exterior durante el último cuarto del siglo xx. (Véase la figura 2.14.) El *Voyager 1* se encontró con Júpiter y Saturno; el *Voyager 2* no sólo encontró a Júpiter y Saturno sino que continuó hacia Urano y Neptuno. El programa Voyager fue enormemente exitoso y la nave espacial *Voyger* continúa recopilando información conforme se aleja del sistema solar. Se espera que los generadores de potencia (reactores nucleares de bajo nivel) en cada nave espacial funcionen hasta al menos el año 2020. La fuente de poder es una muestra de plutonio 238 que, conforme decae, genera calor que se usa para producir electricidad. Durante el lanzamiento de cada nave espacial, su generador produjo alrededor de 470 watts de potencia. Puesto que el plutonio decae, la producción de potencia disminuyó a aproximadamente 335 watts en 1997, casi 20 años después del lanzamiento. Esta potencia se usa para operar el paquete científico, pero si se dirigiese a propulsión, ¿cuánta aceleración produciría en la nave espacial? En la actualidad, el *Voyager 1* viaja con una rapidez de 3.50 AU/año (AU es unidad astronómica), y el *Voyager 2* viaja a 3.15 AU/año. Cada nave espacial pesa 721.9 kg.

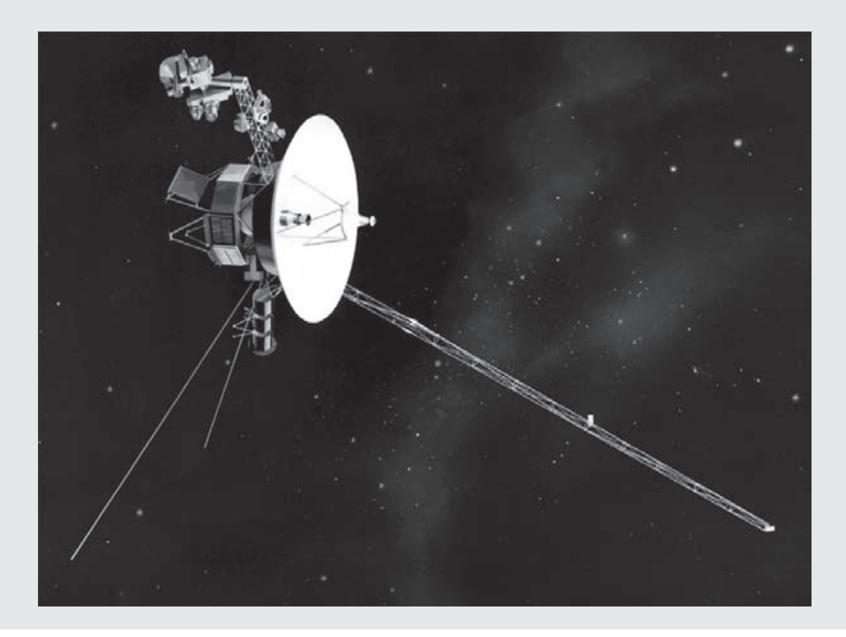


Figura 2.14
Las naves espaciales *Voyager 1* y *Voyager 2* se lanzaron en 1977 y ya abandonaron el sistema solar. (Cortesía de NASA/Jet Propulsion Laboratory.)

1. Establezca el problema.
   Encontrar la aceleración que es posible con la salida de potencia de los generadores de potencia de la nave espacial.
2. Describa las entradas y salidas.

   Entrada

Masa	=	721.9 kg	
Potencia	=	335 watts = 335 J/s	
Rapidez	=	3.50 AU/año	(*Voyager 1*)
Rapidez	=	3.15 AU/año	(*Voyager 2*)

   Salida

   Aceleración de cada nave espacial, en m/s/s
3. Desarrolle un ejemplo a mano.
   Se sabe que

   $$P = m \times a \times v$$

   que se puede reordenar para obtener

   $$a = P/(m \times v)$$

   La parte más difícil de este cálculo será conservar las unidades correctas. Primero, cambie la rapidez a m/s. Para el *Voyager 1*,

   $$v = 3.50 \frac{\text{AU}}{\text{año}} \times \frac{150 \times 10^9 \text{ m}}{\text{AU}} \times \frac{\text{año}}{365 \text{ días}} \times \frac{\text{día}}{24 \text{ horas}} \times \frac{\text{hora}}{3600 \text{ s}} = 16{,}650 \frac{\text{m}}{\text{s}}$$

   Luego se calcula la aceleración:

   $$a = \frac{335 \frac{\text{J}}{\text{s}} \times 1 \frac{\text{kg} \times \text{m}^2}{\text{s}^2 \text{ J}}}{721.9 \text{ kg} \times 16{,}650 \frac{\text{m}}{\text{s}}} = 2.7 \times 10^{-5} \frac{\text{m}}{\text{s}^2}$$

4. Desarrolle una solución MATLAB.

```
clear, clc
%Ejemplo 2.4
%Encuentre la posible aceleración de las naves espaciales
%Voyager 1 y Voyager 2 usando el generador de potencia
%a bordo
format short
m=721.9;      %masa en kg
p=335;        % potencia en watts
v=[3.5 3.15]; % rapidez en AU/año
%Cambia la rapidez a m/s
v=v*150e9/365/24/3600
%Calcula la aceleración
a=p./(m.*v)
```

   Los resultados se imprimen en la ventana de comandos, como se muestra en la figura 2.15.

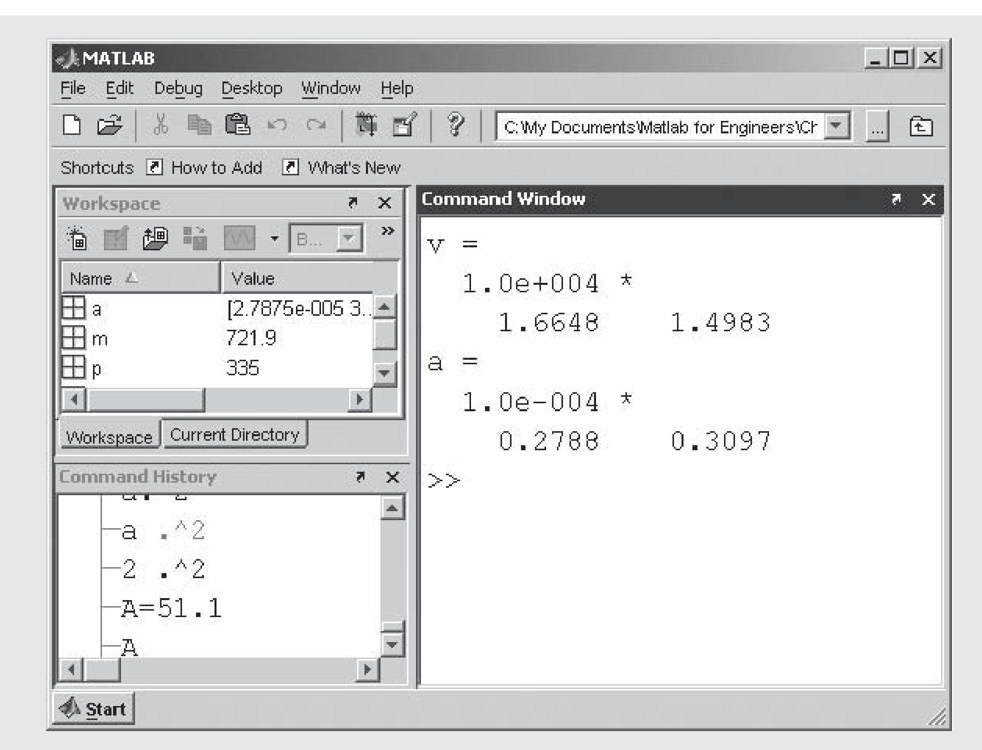


Figura 2.15
Los resultados de la ejecución de un archivo-m se imprimen en la ventana de comandos. Las variables que se crearon se reflejan en el área de trabajo y el archivo-m se cita en la ventana de directorio actual. Los comandos emitidos en el archivo-m no se reflejan en la historia de comandos.

5. Ponga a prueba la solución.
   Compare los resultados MATLAB con los resultados del ejemplo a mano. Note que la rapidez y la aceleración calculadas en el ejemplo a mano coinciden con la solución MATLAB para el *Voyager 1*. La aceleración parece muy pequeña, pero al aplicarse sobre periodos de semanas o meses, tal aceleración puede lograr cambios de velocidad significativos. Por ejemplo, una aceleración constante de 2.8×10^{-5} m/s^2 resulta en un cambio de velocidad de aproximadamente 72 m/s durante un mes:

$$2.8 \times 10^{-5}\ \text{m/s}^2 \times 3600\ \text{s/hora}$$
$$\times\ 24\ \text{horas/día} \times 30\ \text{días/mes} = 72.3\ \text{m/s}$$

   Ahora que tiene un programa MATLAB que funciona, puede usarlo como el punto de partida para otros cálculos más complicados.

RESUMEN

En este capítulo se introdujo la estructura MATLAB básica. El ambiente MATLAB incluye múltiples ventanas, cuatro de las cuales se abren en la vista por defecto:

- Ventana de comandos (command window).
- Ventana de historia de comandos (command history).
- Ventana de área de trabajo (workspace).
- Ventana de directorio actual (current directory).

Además, las ventanas

- de documentos.
- de gráficas.
- de edición.

se abren conforme se requieran durante una sesión MATLAB.

Las variables que se definen en MATLAB siguen las convenciones de nomenclatura computacional comunes:

- Los nombres deben comenzar con una letra.
- Letras, números y guión bajo son los únicos caracteres permitidos.
- Los nombres son sensibles a mayúsculas y minúsculas.
- Los nombres pueden tener cualquier longitud, aunque MATLAB sólo usa los primeros 63 caracteres.
- Algunas palabras clave se reservan para MATLAB y no se pueden usar como nombres de variables.
- MATLAB permite al usuario reasignar nombres de función como nombres de variable, aunque no es aconsejable hacerlo.

La unidad computacional básica en MATLAB es la matriz. Las matrices pueden ser.

- Escalares (matriz 1×1).
- Vectores (matrices $1 \times n$ o $n \times 1$, o una fila o una columna).
- Arreglos bidimensionales ($m \times n$ o $n \times m$).
- Arreglos multidimensionales.

Con frecuencia, las matrices almacenan información numérica, aunque también pueden almacenar otro tipo de información. Los datos se pueden ingresar en una matriz de forma manual o se pueden recuperar de archivos de datos almacenados. Cuando se ingresa manualmente, una matriz se encierra en corchetes, los elementos en una fila se separan mediante comas o espacios, y una nueva fila se indica con un punto y coma:

```
a = [1 2 3 4; 5 6 7 8]
```

Las matrices igualmente espaciadas se pueden generan con el operador dos puntos. Por ende, el comando

```
b = 0:2:10
```

crea una matriz que comienza en 0, termina en 10 y tiene un incremento de 2. Las funciones **linspace** y **logspace** se pueden usar para generar una matriz de longitud especificada a partir de valores de inicio y fin dados, espaciados lineal o logarítmicamente. La función **help** o el menú Help MATLAB se pueden usar para determinar la sintaxis apropiada para estas y otras funciones.

MATLAB sigue el orden algebraico estándar de las operaciones. Los operadores que soporta MATLAB se mencionan en la sección “Resumen MATLAB” de este capítulo.

MATLAB soporta notación estándar (decimal) y científica. También soporta algunas diferentes opciones de despliegue, que también se describen en la sección “Resumen MATLAB”. Sin importar cómo se desplieguen los valores, se almacenan como números de punto flotante de doble precisión.

Las colecciones de comandos MATLAB se pueden guardar en archivos-m script. Las variables MATLAB se pueden guardar o importar de archivos .MAT o .DAT. El formato .MAT es propio de MATLAB y se usa porque almacena datos de manera más eficiente que otros formatos de archivo. El formato .DAT emplea el formato ASCII estándar y se usa cuando los datos creados en MATLAB se compartirán con otros programas.

RESUMEN MATLAB

El siguiente resumen MATLAB cita todos los caracteres especiales, comandos y funciones que se definieron en este capítulo:

Caracteres especiales	
[ ]	forma matrices
( )	se usa en enunciados para agrupar operaciones
	se usa con un nombre de matriz para identificar elementos específicos
,	separa subíndices o elementos de matriz
;	separa filas en una definición de matriz
	suprime salida cuando se usa en comandos
:	se usa para generar matrices
	indica todas las filas o todas las columnas
=	operador asignación: asigna un valor a una ubicación de memoria;
	no es lo mismo que una igualdad
%	indica un comentario en un archivo-m
+	suma escalar y de arreglo
-	resta escalar y de arreglo
*	multiplicación escalar y multiplicación en álgebra matricial
.*	multiplicación de arreglo (multiplicación punto o punto estrella)
/	división escalar y división en álgebra matricial
./	división de arreglo (división punto o punto diagonal)
^	exponenciación escalar y exponenciación de matriz en álgebra matricial
.^	exponenciación de arreglo (potencia punto o punto caret)

Comandos y funciones	
ans	nombre de variable por defecto para resultados de cálculos MATLAB
ascii	indica que los datos se deben guardar en formato ASCII estándar
clc	limpia la ventana de comandos
clear	limpia el área de trabajo
exit	termina MATLAB
format +	establece formato sólo a signos más y menos
format compact	establece formato a forma compacta
format long	establece formato a 14 lugares decimales
format long e	establece formato a notación científica con 14 lugares decimales
format loose	establece formato a la forma no compacta por defecto
format short	establece formato al defecto, 4 lugares decimales
format short e	establece formato a notación científica con 4 lugares decimales
format rat	establece formato a despliegue racional (fraccional)
help	invoca la utilidad de ayuda
linspace	función vector linealmente espaciado
load	carga matrices desde un archivo
logspace	función vector logarítmicamente espaciado
pi	aproximación numérica del valor de π
quit	termina MATLAB
save	guarda variables en un archivo
who	lista variables en memoria
whos	lista variables y sus tamaños

TÉRMINOS CLAVE

archivo-m
área de trabajo
argumentos
arreglo
ASCII
asignación
botón de inicio
directorio actual
editor de arreglo
escalar
función
historia de comandos
matriz
notación científica
operador
prompt (incitador)
script
transponer
vector
ventana de comandos
ventana de documentos
ventana de edición
ventana de gráficas

PROBLEMAS

Inicio

2.1 Prediga el resultado de los siguientes cálculos MATLAB:

```
1 + 3/4
5*6*4/2
5/2*6*4
5^2*3
5^(2*3)
1 + 3 + 5/5 + 3 + 1
(1 + 3 + 5)/(5 + 3 + 1)
```

Verifique sus resultados al ingresar los cálculos en la ventana de comandos.

Uso de variables

2.2 Identifique cuál de cada uno de los siguientes pares es un nombre de variable MATLAB legítimo:

fred	fred!
book_1	book-1
2ndplace	Second_Place
#1	No_1
vel_5	vel.5
tan	while

Pruebe sus respuestas con el uso de **isvarname**; por ejemplo,

```
isvarname fred
```

Recuerde: **isvarname** regresa un 1 si el nombre es válido y 0 si no lo es. Aunque es posible reasignar un nombre de función como nombre de variable, hacerlo no es una buena idea. Use **which** para verificar si los nombres precedentes son nombres de funciones; por ejemplo,

```
which sin
```

¿En qué caso MATLAB le diría que **sin** es un nombre de variable, no un nombre de función?

Operaciones escalares y orden de operaciones

2.3 Cree código MATLAB para realizar los cálculos siguientes:

$$5^2$$

$$\frac{5 + 3}{5 \cdot 6}$$

$$\sqrt{4 + 6^3}$$ (*Sugerencia:* una raíz cuadrada es lo mismo que una potencia ½.)

$$9\frac{6}{12} + 7 \cdot 5^{3+2}$$

$$1 + 5 \cdot 3/6^2 + 2^{2-4} \cdot 1/5.5$$

Compruebe su código al ingresarlo en MATLAB y mediante la realización de los cálculos en su calculadora científica.

2.4 **(a)** El área de un círculo es πr^2. Defina r como 5 y luego encuentre en MATLAB el área de un círculo.

(b) El área superficial de una esfera es $4\pi r^2$. Encuentre el área superficial de una esfera con un radio de 10 pies.

(c) El volumen de una esfera es $4/3\, \pi r^3$. Encuentre el volumen de una esfera con un radio de 2 pies.

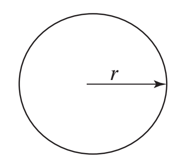
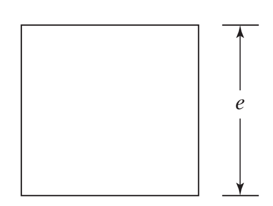


Figura P2.4(a)

2.5 **(a)** El área de un cuadrado es la longitud de arista al cuadrado. (A = arista^2.) Defina la longitud de arista como 5 y luego encuentre en MATLAB el área de un cuadrado.

(b) El área superficial de un cubo es 6 veces la longitud de arista al cuadrado. (AS = 6 × arista^2.) Encuentre el área superficial de un cubo con longitud de arista 10.

(c) El volumen de un cubo es la longitud de arista al cubo. (V = arista^3.) Encuentre el volumen de un cubo con longitud de arista 12.



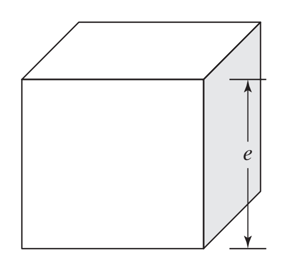


Figura P2.5(a-c)

Operaciones de arreglos

2.6 **(a)** El volumen de un cilindro es $\pi r^2 h$. Defina r como 3 y h como la matriz



```
h = [1, 5, 12]
```

Encuentre el volumen de los cilindros.

(b) El área de un triángulo es ½ la longitud de la base del triángulo, por la altura del triángulo. Defina la base como la matriz

```
b = [ 2, 4, 6]
```

y la altura h como 12, y encuentre el área de los triángulos.

(c) El volumen de cualquier prisma recto es el área de la base del prisma, por la dimensión vertical del prisma. La base del prisma puede ser cualquier forma, por ejemplo,

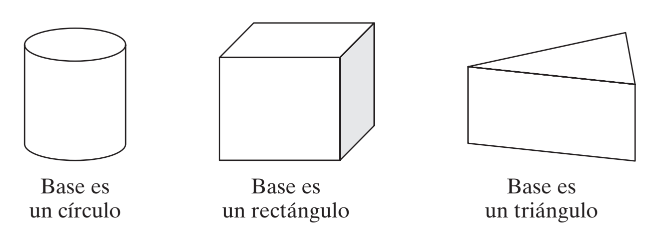


Figura P2.6(c)

Encuentre el volumen de los prismas creados a partir de los triángulos de la parte (b). Suponga que la dimensión vertical de estos prismas es 6.

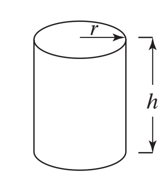


Figura P2.6(a)

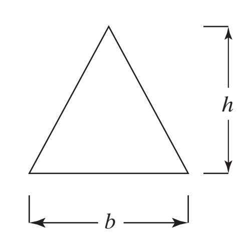


Figura P2.6(b)

2.7 **(a)** Cree un vector igualmente espaciado de valores desde 1 hasta 20 en incrementos de 1.

(b) Cree un vector con valores desde cero hasta 2π en incrementos de $\pi/10$.

(c) Cree un vector que contenga 15 valores, igualmente espaciados entre 4 y 20. (*Sugerencia:* use el comando **linspace**. Si no puede recordar la sintaxis, escriba **help linspace**.)

(d) Cree un vector que contenga 10 valores logarítmicamente espaciados entre 10 y 1000. (*Sugerencia:* use el comando **logspace**.)

2.8 **(a)** Cree una tabla de conversión de pies a metros. Comience la columna pies en 0, incremente en 1 y termine en 10 pies. (Busque el factor de conversión en un manual o en línea.)

(b) Cree una tabla de conversiones de radianes a grados. Comience la columna radianes en 0 e incremente por 0.1π radián, hasta π radianes. (Busque el factor de conversión en un manual o en línea.)

(c) Cree una tabla de conversiones de mi/h a pies/s. Comience la columna mi/h en 0 y termine en 100 mi/h. Imprima 15 valores en su tabla. (Busque el factor de conversión en un manual o en línea.)

(d) La acidez de las soluciones, por lo general, se mide en términos de pH. El pH de una solución se define como $-og_{10}$ de la concentración de iones hidruro. Cree una tabla de conversiones de concentración de ión hidruro a pH, logarítmicamente espaciada, de .001 a .1 mol/litro con 10 valores. Si supone que la concentración de iones hidruro se nombra **H_conc**, la sintaxis para calcular el logaritmo de la concentración es

```
log10(H_conc)
```

2.9 La ecuación general para la distancia que recorre un cuerpo en caída libre (ignorando la fricción del aire) es

$$d = \frac{1}{2}gt^2$$

Suponga que $g = 9.8$ m/s^2. Genere una tabla de tiempo contra distancia recorrida para valores de tiempo de 0 a 100 segundos. Elija un incremento adecuado para su variable tiempo. (*Sugerencia:* tenga cuidado de usar los operadores correctos; ¡t^2 es una operación arreglo!)

2.10 La ley de Newton de la gravitación universal dice que la fuerza ejercida por una partícula sobre otra es

$$F = G\frac{m_1 m_2}{r^2}$$

donde la constante de gravitación universal G tiene el valor experimental de

$$G = 6.673 \times 10^{-11} \text{ N m}^2/\text{kg}^2$$

La masa de cada partícula es m_1 y m_2, respectivamente, y r es la distancia entre las dos partículas. Use la ley de Newton de gravitación universal para encontrar la fuerza que ejerce la Tierra sobre la Luna, si supone que

la masa de la Tierra es aproximadamente 6×10^{24} kg,
la masa de la Luna es aproximadamente 7.4×10^{22} kg, y
la Tierra y la Luna están separadas una distancia promedio de 3.9×10^8 m.

2.11 Se sabe que la Tierra y la Luna no siempre están separadas la misma distancia. Encuentre la fuerza que la Luna ejerce sobre la Tierra para 10 distancias entre 3.8×10^8 m y 4.0×10^8 m.

Despliegue de números

2.12 Cree una matriz **a** igual a [−1/3, 0, 1/3, 2/3] y use cada una de las opciones de formato interno para desplegar los resultados:

formato corto (que es por defecto)
formato largo
formato bank
formato corto e
formato largo e
formato +
formato rat

Cómo guardar trabajo en archivos

2.13
- Cree una matriz llamada G_a_R compuesta de dos columnas, una que represente grados y la otra que represente el valor correspondiente en radianes. Para este ejercicio es válido cualquier conjunto de valores.
- Guarde la matriz a un archivo llamado degrees.dat.
- Una vez guardado el archivo, limpie su área de trabajo y luego cargue los datos del archivo de vuelta a MATLAB.

2.14 Cree un archivo-m script y úselo para hacer los problemas de tarea en este capítulo. Su archivo debe incluir comentarios adecuados para identificar cada problema y describir su proceso de cálculo. No olvide incluir su nombre, la fecha y cualquier otra información que solicite su profesor.



CAPÍTULO 3

Funciones internas de MATLAB

Objetivos

Después de leer este capítulo, el alumno será capaz de

- usar una variedad de funciones matemáticas comunes.
- entender y usar funciones trigonométricas en MATLAB.
- calcular y usar funciones de análisis estadístico y de datos.
- generar matrices uniformes y gaussianas de números aleatorios.
- comprender los límites computaciones de MATLAB.
- reconocer y ser capaz de usar los valores y funciones especiales internos de MATLAB.

INTRODUCCIÓN

La gran mayoría de los cálculos de ingeniería requieren funciones matemáticas muy complicadas, incluidos logaritmos, funciones trigonométricas y funciones de análisis estadístico. MATLAB tiene una extensa librería de funciones internas que le permiten realizar dichos cálculos.

3.1 USO DE FUNCIONES INTERNAS

Muchos de los nombres de las funciones internas de MATLAB son los mismos que los definidos no sólo en el lenguaje de programación C, sino también en FORTRAN. Por ejemplo, para sacar la raíz cuadrada de las variable **x**, se escribe

```
b = sqrt(x);
```

Una de las grandes ventajas de MATLAB es que los argumentos de la función, por lo general, pueden ser escalares o matrices. En el ejemplo, si **x** es un escalar, se regresa un resultado escalar. Por tanto, el enunciado

```
x = 9;
b = sqrt(x)
```

regresa un escalar:

```
b=
     3
```

Sin embargo, la función raíz cuadrada, **sqrt**, también puede aceptar matrices como entrada. En este caso, se calcula la raíz cuadrada de cada elemento, de modo que

```
x = [4, 9, 16];
b = sqrt(x)
```

regresa

```
b =
     2     3     4
```

Idea clave: la mayoría de los nombres de función MATLAB son las mismas que las usadas en otros programas de cómputo.

argumento: entrada a una función

Se puede considerar que todas las funciones tienen tres componentes: nombre, entrada y salida. En el ejemplo precedente, el nombre de la función es **sqrt**, la entrada requerida (también llamada *argumento*) va dentro de los paréntesis y puede ser un escalar o una matriz, y la salida es un valor o valores calculados. En este ejemplo, a la salida se le asignó el nombre de variable **b**.

Algunas funciones requieren múltiples entradas. Por ejemplo, la función residuo (remainder), **rem**, requiere dos entradas: un dividendo y un divisor. Esto se representa como **rem(x,y)**, de modo que

```
rem(10,3)
```

calcula el residuo de 10 dividido entre 3:

```
ans =
      1
```

La función **size** es un ejemplo de una función que regresa dos salidas. Determina el número de filas y columnas en una matriz. Por tanto,

```
d = [1, 2, 3; 4, 5, 6];
f = size(d)
```

regresa la matriz resultante 1 × 2

```
f =
      2     3
```

También puede asignar nombres de variable a cada una de las respuestas al representar el lado izquierdo del enunciado de asignación como una matriz. Por ejemplo,

```
[x,y] = size(d)
```

produce

```
x =
      2
y =
      3
```

anidado: uso de una función como la entrada a otra

También puede crear expresiones complicadas mediante funciones anidadas (nesting). Por ejemplo,

```
g = sqrt(sin(x))
```

encuentra la raíz cuadrada del seno de cualesquier valores almacenados en la matriz llamada **x**. Dado que, anteriormente en esta sección, a **x** se le asignó un valor de 2, el resultado es

```
g =
      0.9536
```

Las funciones anidadas pueden resultar en código MATLAB algo complicado. Asegúrese de incluir los argumentos para cada función dentro de su propio conjunto de paréntesis. Desde luego, su código será más sencillo de leer si descompone las expresiones anidadas en dos enunciados separados. Por tanto,

```
a = sin(x);
g = sqrt(a)
```

da el mismo resultado que **g = sqrt(sin(x))** y es más fácil de seguir.

> **Sugerencia**
>
> Probablemente pueda *suponer* el nombre y sintaxis para muchas funciones MATLAB. Sin embargo, verifique para estar seguro de que la función en la que está interesado funciona de la forma en que supone antes de realizar cálculos importantes.

3.2 USO DE LA AYUDA

MATLAB incluye extensas herramientas de ayuda, lo que es especialmente útil para entender cómo usar las funciones. Existen dos formas de obtener ayuda desde el interior de MATLAB: una función de ayuda de línea de comando (**help**) y un conjunto de documentos HTML disponibles al seleccionar **Help** de la barra de menú o al usar la tecla de función F1, que, por lo general, se ubica en la parte superior de su teclado (o que se encuentra al escribir **helpwin** en la ventana de comandos). También existe un conjunto de documentos de ayuda en línea, disponibles a través del botón Start o el icono Help en la barra de menú. Debe usar ambas opciones de ayuda, pues ellas ofrecen diferente información y pistas acerca de cómo usar una función específica.

Para usar la función de ayuda de línea de comando, escriba **help** en la ventana de comandos:

```
help
```

Aparecerá una lista de temas de ayuda:

Idea clave: use la función help para ayudarse a usar las funciones internas de MATLAB.

```
Temas HELP:

MATLAB\general        – Comandos de propósito general
MATLAB\ops            – Caracteres y operadores
                        especiales
MATLAB\lang           – Constructos de lenguaje
                        de programación
MATLAB\elmat          – Matrices elementales y
                        manipulación de matrices
MATLAB\elfun          – Funciones matemáticas elementales
MATLAB\specfun        – Funciones matemáticas especializadas
Etc., etc.
```

Para obtener ayuda acerca de un tema particular, escriba **help <topic>**. (Recuerde que los paréntesis angulados, < y >, identifican dónde debe escribir su entrada; no se incluyen en el enunciado MATLAB real.)

Por ejemplo, para obtener ayuda acerca de la función **tangent**, escriba

```
help tan
```

Se debe mostrar lo siguiente:

```
TAN    Tangent.
 TAN(X) is the tangent of the elements of X.

 See also ATAN, ATAN2.
```

Para usar la pantalla de ayuda en ventana, seleccione **Help → MATLAB Help** de la barra de menú. Aparecerá una versión en ventana de la lista de ayuda. (Véase la figura 3.1.)

Esta función de ayuda incluye un tutorial MATLAB que encontrará extremadamente útil. La lista en la ventana izquierda es una tabla de contenidos. Note que la tabla de contenidos incluye una liga a una lista de funciones, organizada tanto por categoría como alfabéticamente por nom-

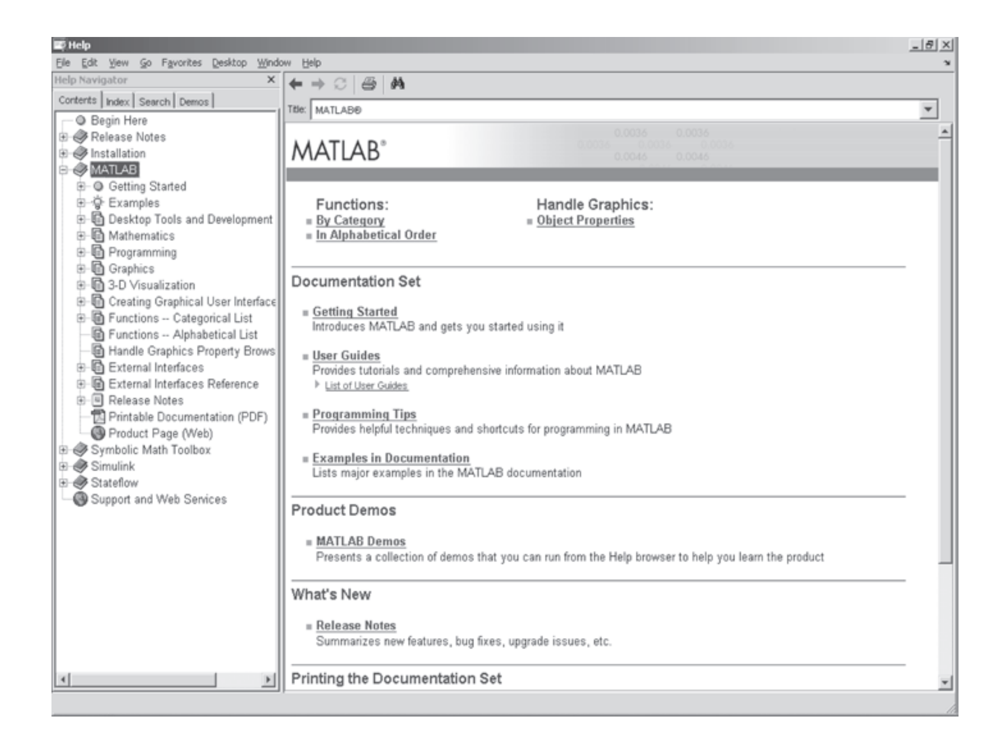


Figura 3.1
El ambiente de ayuda MATLAB.

bre. Puede usar esta liga para encontrar qué funciones MATLAB están disponibles para resolver muchos problemas. Por ejemplo, es posible que quiera redondear un número que calculó. Use la ventana de ayuda MATLAB para determinar si está disponible una función MATLAB adecuada.

Seleccione la liga **MATLAB Functions Listed by Category** (funciones MATLAB listadas por categoría) (véase la figura 3.1) y luego la liga Mathematics (véase la figura 3.2).

Casi a la mitad de la página está la categoría Elementary Math, que menciona rounding (redondeo) como un tema. Siga la liga y encontrará toda una categoría dedicada a las funciones de redondeo. Por ejemplo, **round** redondea al entero más cercano.

También podría haber encontrado la sintaxis para la función round al seleccionar **Functions-Alphabetical List**.

Sugerencia

Puede solicitar la característica de ayuda en ventana al usar el comando **doc**, como en **doc round**.

Ejercicio de práctica 3.1

1. Use el comando help en la ventana de comandos para encontrar la sintaxis adecuada para las siguientes funciones:
   a. cos
   b. sqrt
   c. exp
2. Use la función de ayuda en ventana de la barra de menú para aprender acerca de las funciones en el problema 1.
3. Vaya a la función de ayuda en línea en www.mathworks.com para aprender acerca de las funciones en el problema 1.

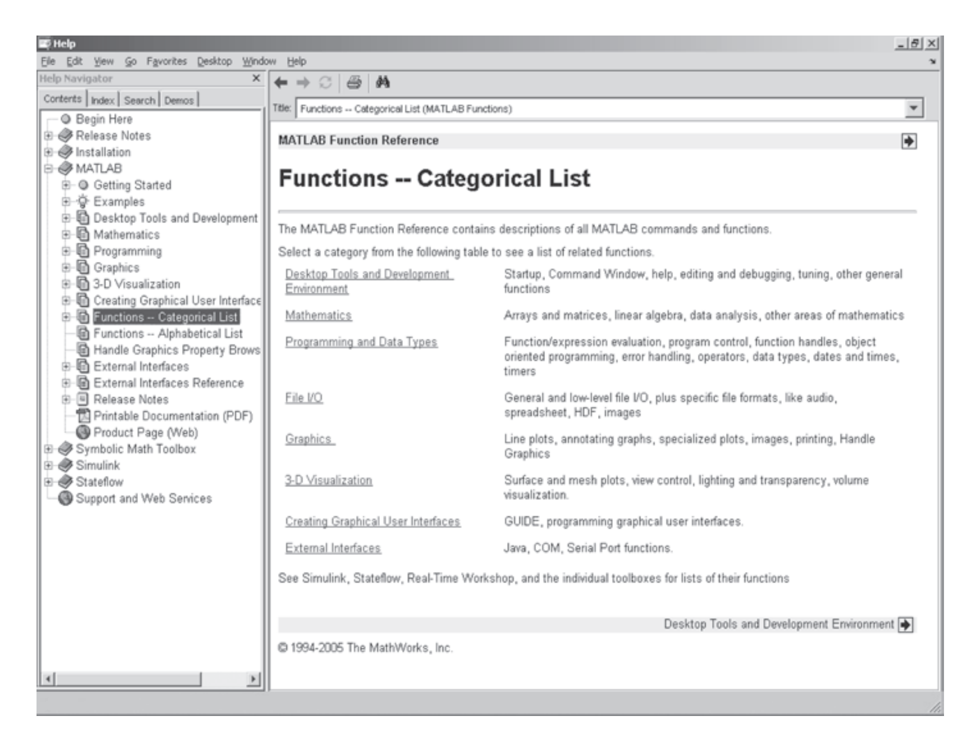


Figura 3.2
Ventana de ayuda para funciones matemáticas.

3.3 FUNCIONES MATEMÁTICAS ELEMENTALES

Las funciones matemáticas elementales incluyen logaritmos, exponenciales, valor absoluto, funciones de redondeo y funciones que se usan en matemáticas discretas.

Idea clave: la mayoría de las funciones aceptan como entrada escalares, vectores o matrices.

3.3.1 Cálculos comunes

Las funciones que se mencionan en la tabla 3.1 aceptan un escalar o una matriz de **x** valores.

Sugerencia

Como regla, la función **log** en todos los lenguajes de cómputo significa el **logaritmo natural**. Aunque no es el estándar en los textos de matemáticas, *es* el estándar en la programación de computadoras. No conocer esta distinción es una fuente de error común, en especial para nuevos usuarios. Si quiere logaritmos a la base 10, necesitará usar la función **log10**. En MATLAB también se incluye una función **log2**, pero los logaritmos a cualquiera otra base necesitan calcularse; no hay una función logaritmo general que permita al usuario ingresar la base.

Ejercicio de práctica 3.2

1. Cree un vector **x** de −2 a +2 con un incremento de 1. Su vector debe ser

```
x = [−2, −1, 0, 1, 2]
```

   a. Encuentre el valor absoluto de cada miembro del vector.
   b. Encuentre la raíz cuadrada de cada miembro del vector.

2. Encuentre la raíz cuadrada de −3 y +3.
   a. Use la función **sqrt**.
   b. Use la función **nthroot**.
   c. Eleve −3 y +3 a la potencia ½.
   ¿Cómo varía el resultado?
3. Cree un vector **x** de −10 a 11 con un incremento de 3.
   a. Encuentre el resultado de **x** dividido entre 2.
   b. Encuentre el residuo de **x** dividido entre 2.
4. Use el vector del problema 3 y encuentre $\mathbf{e}^{\mathbf{x}}$.
5. Use el vector del problema 3.
   a. Encuentre ln(**x**) (el logaritmo natural de **x**).
   b. Encuentre $\log_{10}$(**x**) (el logaritmo común de **x**).
   Explique sus resultados.
6. Use la función **sign** para determinar cuáles de los elementos en el vector **x** son positivos.
7. Cambie el **format** a **rat** y muestre el valor del vector **x** dividido entre 2.

Tabla 3.1 Funciones matemáticas comunes

`abs(x)`	Encuentra el valor absoluto de **x**	`abs(-3)` `ans = 3`
`sqrt(x)`	Encuentra la raíz cuadrada de **x**	`sqrt(85)` `ans = 9.2195`

`nthroot(x,n)`	Encuentra la *n*-ésima raíz real de **x**. Esta función no regresará resultados complejos. Por tanto, **(−2)^(1/3)** no regresa el mismo resultado, aunque ambas respuestas son legítimas raíces cúbicas de −2	`nthroot(-2,3)`
		`ans =`
		`-1.2599`
		
		`(-2)^(1/3)`
		`ans =`
		`0.6300 + 1.0911i`

`sign(x)`	Regresa un valor de −1 si **x** es menor que cero, un valor de 0 si **x** es igual a cero y un valor de +1 si **x** es mayor que cero	`sign( -8)` `ans = -1`
`rem(x,y)`	Calcula el residuo de **x/y**	`rem(25,4)` `ans = 1`
`exp(x)`	Calcula el valor de e^x, donde *e* es la base para logaritmos naturales, o aproximadamente 2.7183	`exp(10)` `ans = 2.2026e+004`
`log(x)`	Calcula ln(**x**), el logaritmo natural de **x** (a la base *e*)	`log(10)` `ans = 2.3026`
`log10(x)`	Calcula $\log_{10}$(**x**), el logaritmo común de **x** (a la base 10)	`log10(10)` `ans = 1`

Sugerencia

La notación matemática y la sintaxis MATLAB para elevar e a una potencia no son iguales. Para elevar e a la tercera potencia, la notación matemática sería e^3. Sin embargo, la sintaxis MATLAB es **exp(3)**. A veces los estudiantes también confunden la sintaxis de la notación científica con los exponenciales. El número 5e3 se debe interpretar como 5×10^3.

EJEMPLO 3.1

Uso de la ecuación Clausius–Clapeyron

Los meteorólogos estudian la atmósfera con la intención de comprender y a final de cuentas predecir el clima. (Véase la figura 3.3.) La predicción del clima es un proceso complicado, incluso con los mejores datos. Los meteorólogos estudian química, física, termodinámica y geografía, además de cursos especializados acerca de la atmósfera.

Una ecuación que usan los meteorólogos es la ecuación Clausius-Clapeyron, que, por lo general, se introduce en las clases de química y se examina con más detalle en clases de termodinámica avanzada.

En meteorología, la ecuación Clausius-Clapeyron se emplea para determinar la relación entre presión de vapor de agua de saturación y la temperatura atmosférica. La presión de vapor de agua de saturación se puede usar para calcular la humedad relativa, un componente importante de la predicción del clima, cuando se conoce la verdadera presión parcial del agua en el aire.

La ecuación Clausius-Clapeyron es

$$\ln\left(\frac{P^0}{6.11}\right) = \left(\frac{\Delta H_v}{R_{\text{air}}}\right) * \left(\frac{1}{273} - \frac{1}{T}\right)$$

donde

P^0	=	presión de vapor de saturación para el agua, en mbar, a temperatura T,
ΔH_v	=	calor latente de vaporización para el agua, 2.453×10^6 J/kg,
R_{air}	=	constante de gas para el aire húmedo, 461 J/kg, y
T	=	temperatura en grados kelvin (K).

Es raro que las temperaturas en la superficie de la Tierra sean menores que −60 °F o mayores que 120 °F. Use la ecuación Clausius-Clapeyron para encontrar la presión de vapor de saturación para temperaturas en este rango. Presente sus resultados como una tabla de temperaturas Fahrenheit y presiones de vapor de saturación.

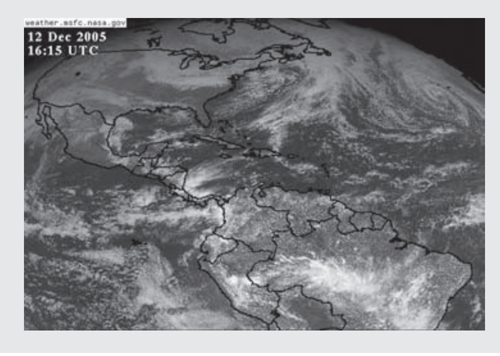


Figura 3.3
Vista del clima de la Tierra desde el espacio. (Cortesía de NASA/Jet Propulsion Laboratory.)

1. Establezca el problema.
   Encontrar la presión de vapor de saturación a temperaturas de −60 °F a 120 °F, con la ecuación Clausius-Clapeyron.
2. Describa las entradas y salidas.

   Entrada

$$\Delta H_v = 2.453 \times 10^6 \text{ J/kg}$$
$$R_{\text{air}} = 461 \text{ J/kg}$$
$$T = -60 \text{ °F a } 120 \text{ °F}$$

   Dado que no se especificó el número de valores de temperatura, se elegirá recalcular cada 10 °F.

   Salida

   Presiones de vapor de saturación
3. Desarrolle un ejemplo a mano.
   La ecuación Clausius-Clapeyron requiere que todas las variables tengan unidades consistentes. Esto significa que la temperatura (T) necesita estar en grados kelvin. Para cambiar grados Fahrenheit a kelvin, se usa la ecuación de conversión

$$T_k = \frac{(T_f + 459.6)}{1.8}$$

   (Existen muchos lugares donde encontrar conversiones de unidades. Internet es una fuente, como los libros de ciencia e ingeniería.)

   Ahora se requiere resolver la ecuación Clausius-Clapeyron para la presión de vapor de saturación P^0. Se tiene

$$\ln\left(\frac{P^0}{6.11}\right) = \left(\frac{\Delta H_v}{R_{\text{air}}}\right) \times \left(\frac{1}{273} - \frac{1}{T}\right)$$
$$P^0 = 6.11 \times \exp\left(\left(\frac{\Delta H_v}{R_{\text{air}}}\right) \times \left(\frac{1}{273} - \frac{1}{T}\right)\right)$$

   A continuación, se resuelve para una temperatura (por ejemplo, $T = 0$ °F) para obtener

$$T = \frac{(0 + 459.6)}{1.8} = 255.3333$$

   Finalmente, se sustituyen valores para obtener

$$P^0 = 6.11 \times \exp\left(\left(\frac{2.453 \times 10^6}{461}\right) \times \left(\frac{1}{273} - \frac{1}{255.3333}\right)\right) = 1.5836 \text{ mbar}$$

4. Desarrolle una solución MATLAB.
   Cree la solución MATLAB en un archivo-m y luego córralo en el ambiente de comandos:

```
%Ejemplo 3.1
%Con la ecuación Clausius-Clapeyron, encontrar la
%presión de vapor de saturación para agua a diferentes
%temperaturas
```

```
%
TF=[-60:10:120];      %Define la matriz temp en F
TK=(TF + 459.6)/1.8;  %Convierte temp a K
Delta_H=2.45e6;       %Define calor latente de vaporización
R_air = 461;          %Define constante de gas ideal para aire
%
%Calcula las presiones de vapor
Vapor_Pressure = 6.11*exp((Delta_H/R_air)*(1/273 - 1./TK));
%Muestra los resultados en una tabla
my_results = [TF',Vapor_Pressure']
```

Cuando usted crea un programa MATLAB, es buena idea comentar libremente (líneas que comienzan con %). Esto hace que su programa sea más comprensible para otros y puede hacer que usted lo “depure” más fácilmente. Note que la mayoría de las líneas de código terminan con un punto y coma, lo que suprime la salida. En consecuencia, la única información que se despliega en la ventana de comandos es la tabla **my_results**:

```
my_results =
  -60.0000    0.0698
  -50.0000    0.1252
  -40.0000    0.2184
  -30.0000    0.3714
  -20.0000    0.6163
  -10.0000    1.0000
         0    1.5888
   10.0000    2.4749
   20.0000    3.7847
   30.0000    5.6880
   40.0000    8.4102
   50.0000   12.2458
   60.0000   17.5747
   70.0000   24.8807
   80.0000   34.7729
   90.0000   48.0098
  100.0000   65.5257
  110.0000   88.4608
  120.0000  118.1931
```

5. Ponga a prueba la solución.
   Compare la solución MATLAB con $T = 0$ °F con la solución a mano:

   Solución a mano: $P^0 = 1.5888$ mbar
   Solución MATLAB: $P^0 = 1.5888$ mbar

   La ecuación Clausius-Clapeyron se puede usar para más que sólo problemas de humedad. Al cambiar los valores de ΔH y R podrá generalizar el programa para tratar con cualquier vapor de condensación.

3.3.2 Funciones de redondeo

MATLAB contiene funciones para algunas diferentes técnicas de redondeo (tabla 3.2). Probablemente usted esté más familiarizado con el entero más cercano; sin embargo, tal vez quiera

Tabla 3.2 Funciones de redondeo

`round(x)`	Redondea **x** al entero más cercano	`round(8.6)` `ans = 9`

`fix(x)`	Redondea (o trunca) **x** al entero más cercano hacia cero. Note que con esta función 8.6 se trunca a 8, no a 9	`fix(8.6)`
		`ans = 8`
		
		`fix(-8.6)`
		`ans = -8`

`floor(x)`	Redondea **x** al entero más cercano hacia infinito negativo	`floor(-8.6)` `ans = -9`
`ceil(x)`	Redondea **x** al entero más cercano hacia infinito positivo	`ceil(-8.6)` `ans = -8`

redondear arriba o abajo, dependiendo de la situación. Por ejemplo, suponga que quiere comprar manzanas en la tienda. Las manzanas cuestan \$0.52 la pieza. Usted tiene \$5.00. ¿Cuántas manzanas puede comprar? Matemáticamente,

$$\frac{\$5.00}{\$0.52/\text{manzana}} = 9.6154 \text{ manzanas}$$

Pero, obviamente, usted no puede comprar parte de una manzana, y la tienda no le permitirá redondear al número más cercano de manzanas. En vez de ello, requiere redondear hacia abajo. La función MATLAB que logra esto es **fix**. Por tanto,

```
fix(5/0.52)
```

regresa el número máximo de manzanas que puede comprar:

```
ans =
  9
```

3.3.3 Matemáticas discretas

MATLAB incluye funciones para factorizar números, encontrar denominadores y múltiplos comunes, calcular factoriales y explorar números primos (tabla 3.3). Todas estas funciones requieren escalares enteros como entrada. Además, MATLAB incluye la función **rats**, que expresa un número punto flotante como un número racional, esto es, una fracción. Las matemáticas discretas son las matemáticas de números enteros.

Ejercicio de práctica 3.3

1. Factorice el número 322.
2. Encuentre el máximo común denominador de 322 y 6.
3. ¿322 es número primo?
4. ¿Cuántos primos existen entre 0 y 322?
5. Aproxime π como número racional.
6. Encuentre 10! (10 factorial).

3.4 FUNCIONES TRIGONOMÉTRICAS

MATLAB incluye un conjunto completo de las funciones trigonométricas estándar y las funciones trigonométricas hiperbólicas. La mayoría de estas funciones suponen que los ángulos

Tabla 3.3 Funciones que se usan en matemáticas discretas

factor(x)	Encuentra los factores primos de **x**	**factor(12)** **ans =** **2 2 3**
gcd(x,y)	Encuentra el máximo común denominador de **x** y **y**	**gcd(10,15)** **ans =** **5**
lcm(x,y)	Encuentra el mínimo común múltiplo de **x** y **y**	**lcm(2,5)** **ans =** **10** **lcm(2,10)** **ans =** **10**
rats(x)	Representa **x** como fracción	**rats(1.5)** **ans =** **3/2**
factorial(x)	Encuentra el valor de **x** factorial (x!). Un factorial es el producto de todos los enteros menores que *x*. Por ejemplo, $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$	**factorial(6)** **ans =** **720**
primes(x)	Encuentra todos los números primos menores que **x**	**primes(10)** **ans =** **2 3 5 7**
isprime(x)	Verifica para ver si **x** es un número primo. Si lo es, la función regresa 1; si no lo es, regresa 0	**isprime(7)** **ans =** **1** **isprime(10)** **ans =** **0**

se expresan en radianes. Para convertir radianes a grados o grados a radianes, se necesita sacar ventaja del hecho de que π radianes es igual a 180 grados:

$$\text{grados} = \text{radianes}\left(\frac{180}{\pi}\right) \quad \text{y} \quad \text{radianes} = \text{grados}\left(\frac{\pi}{180}\right)$$

El código MATLAB que realiza estas conversiones es

Idea clave: la mayoría de las funciones trigonométricas requieren entrada en radianes.

```
degrees = radians*180/pi;
radians = degrees * pi/180;
```

Para realizar estos cálculos, es necesario el valor de π, de modo que una constante, **pi**, es interna a MATLAB. Sin embargo, dado que π no se puede expresar como número en punto flotante, la constante **pi** en MATLAB sólo es una aproximación de la cantidad matemática π. Usualmente esto no es importante; sin embargo, puede notar algunos resultados sorprendentes. Por ejemplo, para

```
sin(pi)
ans =
       1.2246e-016
```

cuando usted espera una respuesta de cero.

Puede acceder a la función help desde la barra de menú para obtener una lista completa de funciones trigonométricas disponibles en MATLAB. La tabla 3.4 muestra algunas de las más comunes.

Tabla 3.4 Funciones trigonométricas

`sin(x)`	Encuentra el seno de **x** cuando **x** se expresa en radianes	`sin(0)` `ans = 0`
`cos(x)`	Encuentra el coseno de **x** cuando **x** se expresa en radianes	`cos(pi)` `ans = -1`
`tan(x)`	Encuentra la tangente de **x** cuando **x** se expresa en radianes	`tan(pi)` `ans = -1.2246e-016`
`asin(x)`	Encuentra el arcoseno, o seno inverso, de **x**, donde **x** debe estar entre −1 y 1. La función regresa un ángulo en radianes entre $\pi/2$ y $-\pi/2$	`asin(-1)` `ans = -1.5708`
`sinh(x)`	Encuentra el seno hiperbólico de **x** cuando **x** se expresa en radianes	`sinh(pi)` `ans = 11.5487`
`asinh(x)`	Encuentra el seno hiperbólico inverso de **x**	`asinh(1)` `ans = 0.8814`
`sind(x)`	Encuentra el seno de **x** cuando **x** se expresa en grados	`sind(90)` `ans = 1`
`asind(x)`	Encuentra el seno inverso de **x** y reporta el resultado en grados	`asind(90)` `ans = 1`

Sugerencia

Con frecuencia, los textos de matemáticas usan la notación $\text{sen}^{-1}(x)$ para indicar una función seno inverso, también llamada arcoseno. Usualmente, los estudiantes se confunden con esta notación e intentan crear código MATLAB paralelo. Sin embargo, note que

a = sin^−1(*x*)

no es un enunciado MATLAB válido, pero, en vez de ello, debe ser

a = asin(x)

Ejercicio de práctica 3.4

Calcule lo siguiente (recuerde que la notación matemática no necesariamente es la misma que la notación MATLAB):

1. $\text{sen}(2\theta)$ para $\theta = 3\pi$.
2. $\cos(\theta)$ para $0 \leq \theta \leq 2\pi$; sea θ que cambia en pasos de 0.2π.
3. $\text{sen}^{-1}(1)$.
4. $\cos^{-1}(x)$ para $-1 \leq x \leq 1$; sea x que cambia en pasos de 0.2.
5. Encuentre el coseno de 45°.
   a. Convierta el ángulo de grados a radianes y luego use la función **cos**.
   b. Use la función **cosd**.
6. Encuentre el ángulo cuyo seno es 0.5. ¿Su respuesta está en grados o radianes?
7. Encuentre la cosecante de 60 grados. Es posible que tenga que usar la función help para encontrar la sintaxis adecuada.



Uso de las funciones trigonométricas

Uno de los cálculos básicos en ingeniería es encontrar la fuerza resultante sobre un objeto que se empuja o jala en múltiples direcciones. Sumar fuerzas es el cálculo principal que se realiza en clases de estática y dinámica. Considere un globo sobre el que actúan las fuerzas que se muestran en la figura 3.4.

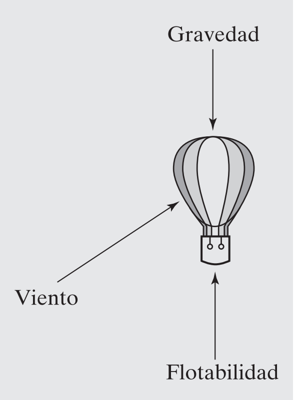


Figura 3.4
Equilibrio de fuerzas sobre un globo.

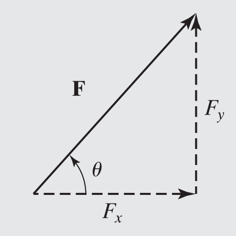
Para encontrar la fuerza neta que actúa sobre el globo, se necesita sumar la fuerza debida a la gravedad, la fuerza debida a la flotabilidad y la fuerza debida al viento. Un enfoque es encontrar la fuerza en la dirección x y la fuerza en la dirección y para cada fuerza individual y luego recombinarlas en un resultado final.

La fuerza en las direcciones x y y se pueden encontrar a partir de trigonometría:

F = fuerza total
F_x = fuerza en la dirección x
F_y = fuerza en la dirección y

A partir de la trigonometría se sabe que el seno es el lado opuesto sobre la hipotenusa, de modo que

$$\text{sen}(\theta) = F_y/F$$



y, por tanto,

$$F_y = F\ \text{sen}(\theta)$$

De igual modo, dado que el coseno es el lado adyacente sobre la hipotenusa,

$$F_x = F\cos(\theta)$$

Se pueden sumar todas las fuerzas en la dirección x y todas las fuerzas en la dirección y y usar estos totales para encontrar la fuerza resultante:

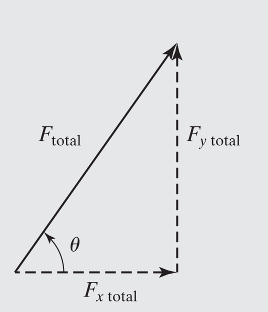
$$F_{x\,\text{total}} = \Sigma F_{xi} \qquad F_{y\,\text{total}} = \Sigma F_{yi}$$

Para encontrar la magnitud y el ángulo de F_{total}, se usa de nuevo la trigonometría. La tangente es el lado opuesto sobre el lado adyacente. Por tanto,

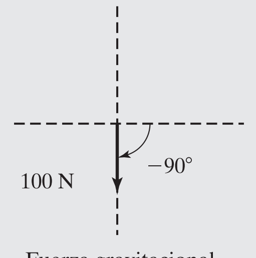
$$\tan(\theta) = \frac{F_{y\,\text{total}}}{F_{x\,\text{total}}}$$

Se usa una tangente inversa para escribir

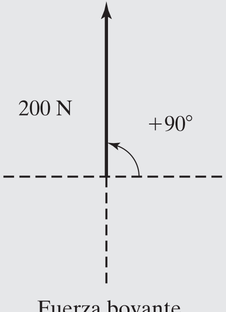
$$\theta = \tan^{-1}\left(\frac{F_{y\,\text{total}}}{F_{x\,\text{total}}}\right)$$



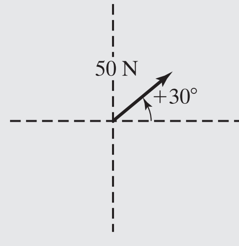
(La tangente inversa también se llama *arcotangente*; la verá en su calculadora científica como atan.)



Fuerza gravitacional



Fuerza boyante



Fuerza del viento

Una vez conocido θ se puede encontrar F_{total} o con seno o con coseno. Se tiene

$$F_{x\ \text{total}} = F_{\text{total}} \cos(\theta)$$

y al reordenar términos se tiene

$$F_{\text{total}} = \frac{F_{x\ \text{total}}}{\cos(\theta)}$$

Ahora considere de nuevo el globo que se muestra en la figura 3.4. Suponga que la fuerza debida a la gravedad en este globo particular es de 100 N, dirigida hacia abajo. Suponga aún más que la fuerza boyante es de 200 N, dirigida hacia arriba. Finalmente, suponga que el viento empuja sobre el globo con una fuerza de 50 N, en un ángulo de 30 grados desde la horizontal.

Encuentre la fuerza resultante sobre el globo.

1. Establezca el problema.
   Encuentre la fuerza resultante sobre un globo. Considere las fuerzas debidas a gravedad, flotabilidad y el viento.
2. Describa las entradas y salidas.

   Entrada

Fuerza	Magnitud	Dirección
Gravedad	100 N	−90 grados
Flotabilidad	200 N	+90 grados
Viento	50 N	+30 grados

Salida

Se necesita encontrar tanto la magnitud como la dirección de la fuerza resultante

3. Desarrolle un ejemplo a mano.
   Encuentre primero los componentes x y y de cada fuerza y sume los componentes:

Fuerza	Componente horizontal	Componente vertical
Gravedad	$F_x = F\cos(\theta)$	$F_y = F \text{ sen } (\theta)$
	$F_x = 100\cos(-90°) = 0$ N	$F_y = 100 \text{ sen } (-90°) = -100$ N
Flotabilidad	$F_x = F\cos(\theta)$	$F_y = F \text{ sen } (\theta)$
	$F_x = 200\cos(+90°) = 0$ N	$F_y = 200 \text{ sen } (+90°) = +200$ N
Viento	$F_x = F\cos(\theta)$	$F_y = F \text{ sen } (\theta)$
	$F_x = 50\cos(+30°) = 43.301$ N	$F_y = 50 \text{ sen } (+30°) = +25$ N
Suma	$F_{x\ \text{total}} = 0 + 0 + 43.301$	$F_{y\ \text{total}} = -100 + 200 + 25$
	$= 43.301$ N	$= 125$ N

Encuentre el ángulo resultante:

$$\theta = \tan^{-1}\left(\frac{F_{y\ \text{total}}}{F_{x\ \text{total}}}\right)$$

$$\theta = \tan^{-1}\frac{125}{43.301} = 70.89°$$

Encuentre la magnitud de la fuerza total:

$$F_{\text{total}} = \frac{F_{x\ \text{total}}}{\cos(\theta)}$$

$$F_{\text{total}} = \frac{43.301}{\cos(70.89°)} = 132.29\ \text{N}$$

4. Desarrolle una solución MATLAB.
   Una solución es:

```
%Ejemplo 3_2
clear, clc
%Define la entrada
F =[100, 200, 50];
theta = [-90, +90, +30];
%Convierte ángulos a radianes
theta = theta*pi/180;
%Encuentra los componentes x
FX = F.*cos(theta);
%Suma los componentes x
FXtotal = sum(FX);
%Encuentra y suma los componentes y en el mismo paso
FYtotal = sum(F.*sin(theta));
%Encuentra el ángulo resultante en radianes
result_angle = atan(FYtotal/FXtotal);
%Encuentra el ángulo resultante en grados
result_degrees = result_angle*180/pi
%Encuentra la magnitud de la fuerza resultante
Ftotal = FXtotal/cos(result_angle)
```

   lo que regresa

```
result_degrees =
       70.8934
Ftotal =
       132.2876
```

   Note que los valores para la fuerza y el ángulo se ingresaron en un arreglo. Esto hace la solución más general. Note también que los ángulos se convirtieron a radianes. En la lista del programa, se suprimió la salida de todos los cálculos, salvo el final. Sin embargo, mientras se desarrolló el programa, se dejaron los puntos y coma de modo que se pudieran observar los resultados intermedios.

5. Ponga a prueba la solución.
   Compare la solución MATLAB con la solución a mano. Ahora que ya sabe que funciona, puede usar el programa para encontrar la resultante de múltiples fuerzas. Sólo agregue la información adicional a las definiciones del vector fuerza **F** y el ángulo del vector **theta**. Note que se supuso un mundo bidimensional en este ejemplo, pero sería fácil extender la solución a fuerzas en tres dimensiones.

3.5 FUNCIONES DE ANÁLISIS DE DATOS

Analizar datos estadísticos en MATLAB es particularmente sencillo, en parte porque todos los conjuntos de datos se pueden representar mediante una sola matriz y en parte debido a la gran cantidad de funciones internas de análisis de datos.

3.5.1 Máximo y mínimo

La tabla 3.5 menciona las funciones que encuentran el mínimo y el máximo en un conjunto de datos y el elemento en el que ocurren dichos valores.

Tabla 3.5 Máximos y mínimos

`max(x)`	Encuentra el valor más grande en un **vector x**. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, el valor máximo es 5	`x=[1, 5, 3];`
		`max(x)`
		`ans =`
		`5`

	Crea un vector fila que contiene el elemento máximo de cada columna de una **matriz x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, entonces el valor máximo en la columna 1 es 2, el valor máximo en la columna 2 es 5 y el valor máximo en la columna 3 es 6	`x=[1, 5, 3; 2, 4, 6];`
		`max(x)`
		`ans =`
		`2 5 6`

`[a,b]=max(x)`	Encuentra tanto el valor más grande en un **vector x** y su ubicación en el vector **x**. Para $x = [1 \quad 5 \quad 3]$, el valor máximo se llama **a** y se encuentra que es 5. La ubicación del valor máximo es el elemento 2 y se llama **b**	`x=[1,5,3];`
		`[a,b]=max(x)`
		`a =`
		`5`
		`b =`
		`2`

	Crea un vector fila que contiene el elemento máximo de cada columna de una matriz *x* y regresa un vector fila con la ubicación del máximo en cada columna de la matriz **x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, entonces el valor máximo en la columna 1 es 2, el valor máximo en la columna 2 es 5 y el valor máximo en la columna 3 es 6. Estos máximos ocurren en la fila 2, fila 1 y fila 2, respectivamente	`x=[1, 5, 3; 2, 4, 6];`
		`[a,b]=max(x)`
		`a =`
		`2 5 6`
		`b =`
		`2 1 2`

`max(x,y)`	Crea una matriz del mismo tamaño que **x** y **y**. (Tanto **x** como **y** deben tener el mismo número de filas y columnas.) Cada elemento en la matriz resultante contiene el valor máximo de las posiciones correspondientes en **x** y **y**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$ y $y = \begin{bmatrix} 10 & 2 & 4 \\ 1 & 8 & 7 \end{bmatrix}$, entonces la matriz resultante será ans $= \begin{bmatrix} 10 & 5 & 4 \\ 2 & 8 & 7 \end{bmatrix}$	`x=[1, 5, 3; 2, 4, 6];`
		`y=[10,2,4; 1, 8, 7];`
		`max(x,y)`
		`ans =`
		`10 5 4`
		`2 8 7`

`min(x)`	Encuentra el valor más pequeño en un **vector x**. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, el valor mínimo es 1	`x=[1, 5, 3];`
		`min(x)`
		`ans =`
		`1`

	Crea un vector fila que contiene el elemento mínimo de cada columna de una **matriz x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, entonces el valor mínimo en la columna 1 es 1, el valor mínimo en la columna 2 es 4 y el valor mínimo en la columna 3 es 3	`x=[1, 5, 3; 2, 4, 6];`
		`min(x)`
		`ans =`
		`1 4 3`

`[a,b]=min(x)`	Encuentra tanto el valor más pequeño en un **vector x** y su ubicación en el vector **x**. Para $x = [1\ 5\ 3]$, el valor mínimo se llama **a** y se encuentra que es 1. La ubicación del valor mínimo es el elemento 1 y se llama **b**	`x=[1,5,3];`
		`[a,b]=min(x)`
		`a =`
		`1`
		`b =`
		`1`

	Crea un vector fila que contiene el elemento mínimo de cada columna de una matriz **x** y regresa un vector fila con la ubicación del mínimo en cada columna de la matriz **x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, entonces el valor mínimo en la columna 1 es 1, el valor mínimo en la columna 2 es 4 y el valor mínimo en la columna 3 es 3. Estos mínimos ocurren en la fila 1, fila 2 y fila 1, respectivamente	`x=[1, 5, 3; 2, 4, 6];`
		`[a,b]=min(x)`
		`a =`
		`1 4 3`
		`b =`
		`1 2 1`

`min(x,y)`	Crea una matriz del mismo tamaño que **x** y **y**. (Tanto **x** como **y** deben tener el mismo número de filas y columnas.) Cada elemento en la matriz resultante contiene el valor mínimo de las posiciones correspondientes en **x** y **y**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$ y $y = \begin{bmatrix} 10 & 2 & 4 \\ 1 & 8 & 7 \end{bmatrix}$, entonces la matriz resultante será ans $= \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \end{bmatrix}$	`x=[1, 5, 3; 2, 4, 6];`
		`y=[10,2,4; 1, 8, 7];`
		`min(x,y)`
		`ans =`
		`1 2 3`
		`1 4 6`

Sugerencia

Todas las funciones en esta sección funcionan sobre las *columnas* en matrices bidimensionales. Si su análisis de datos requiere que evalúe datos en filas, los datos se deben transponer. (En otras palabras, las filas se deben volver columnas y las columnas se deben volver filas.) El operador transpuesto es un una comilla sola ('). Por ejemplo, si quiere encontrar el valor máximo en cada *fila* de la matriz

$$x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$$

use el comando

```
max(x')
```

que regresa

```
ans=
    5    6
```

Ejercicio de práctica 3.5

Considere la siguiente matriz:

$$x = \begin{bmatrix} 4 & 90 & 85 & 75 \\ 2 & 55 & 65 & 75 \\ 3 & 78 & 82 & 79 \\ 1 & 84 & 92 & 93 \end{bmatrix}$$

1. ¿Cuál es el valor máximo en cada columna?
2. ¿En cuál fila se presenta dicho máximo?
3. ¿Cuál es el valor máximo en cada fila? (Tendrá que transponer la matriz para responder esta pregunta.)
4. ¿En cuál columna ocurre el máximo?
5. ¿Cuál es el valor máximo en toda la tabla?

3.5.2 Media y mediana

media: el promedio de todos los valores en el conjunto de datos

mediana: el valor medio en un conjunto de datos

Existen muchas formas de encontrar el valor "promedio" en un conjunto de datos. En estadística, la **media** de un grupo de valores probablemente es lo que la mayoría llamaría el promedio. La media es la suma de todos los valores, divididos por el número total de valores. Otro tipo de promedio es la **mediana**, o el valor medio. Existe un número igual de valores tanto más grandes como más pequeños que la mediana. MATLAB proporciona funciones para encontrar tanto la media como la mediana, como se muestra en tabla 3.6.

Tabla 3.6 Promedios

`mean(x)`	Calcula el valor medio (o valor promedio) de un **vector x**. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, el valor medio es 3	`x=[1, 5, 3];`
		`mean(x)`
		`ans =`
		`  3.0000`

	Regresa un vector fila que contiene el valor medio de cada columna de una **matriz x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, entonces el valor medio de la columna 1 es 1.5, el valor medio de la columna 2 es 4.5 y el valor medio de la columna 3 es 4.5	`x=[1, 5, 3; 2, 4, 6];`
		`mean(x)`
		`ans =`
		`  1.5    4.5    4.5`

`median(x)`	Encuentra la mediana de los elementos de un **vector x**. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, el valor mediana es 3	`x=[1, 5, 3];`
		`median(x)`
		`ans =`
		`  3`

	Regresa un vector fila que contiene el valor mediana de cada columna de una **matriz x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \\ 3 & 8 & 4 \end{bmatrix}$, entonces el valor mediana de la columna 1 es 2, el valor mediana de la columna 2 es 5 y el valor mediana de la columna 3 es 4	`x=[1, 5, 3;`
		`2, 4, 6;`
		`3,8,4];`
		`median(x)`
		`ans =`
		`  2    5    4`

Ejercicio de práctica 3.6

Considere la siguiente matriz:

$$x = \begin{bmatrix} 4 & 90 & 85 & 75 \\ 2 & 55 & 65 & 75 \\ 3 & 78 & 82 & 79 \\ 1 & 84 & 92 & 93 \end{bmatrix}$$

1. ¿Cuál es el valor medio en cada columna?
2. ¿Cuál es la mediana para cada columna?
3. ¿Cuál es el valor medio en cada fila?
4. ¿Cuál es la mediana para cada fila? ¿Cuál es la mediana para toda la matriz?

3.5.3 Sumas y productos

Con frecuencia es útil sumar todos los elementos en una matriz o multiplicar todos los elementos juntos. MATLAB proporciona algunas funciones para calcular tanto sumas como productos, como se muestra en la tabla 3.7.

Tabla 3.7 Sumas y productos

`sum(x)`	Suma los elementos en el **vector x**. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, la suma es 9	`x=[1, 5, 3];`
		`sum(x)`
		`ans =`
		`9`

	Calcula un vector fila que contiene la suma de los elementos en cada columna de una **matriz x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, entonces
	la suma de la columna 1 es 3,
	la suma de la columna 2 es 9 y
	la suma de la columna 3 es 9
		`x=[1, 5, 3; 2, 4, 6];`
		`sum(x)`
		`ans =`
		`3 9 9`

`prod(x)`	Calcula el producto de los elementos de un **vector x**. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, el producto es 15	`x=[1, 5, 3];`
		`prod(x)`
		`ans =`
		`15`

	Calcula un vector fila que contiene el producto de los elementos en cada columna de una **matriz x**.
	Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, entonces
	el producto de la columna 1 es 2,
	el producto de la columna 2 es 20 y
	el producto de la columna 3 es 18
		`x=[1, 5, 3; 2, 4, 6];`
		`prod(x)`
		`ans =`
		`2 20 18`

(*Continúa*)

Tabla 3.7 (continuación)

`cumsum(x)`	Calcula un vector del mismo tamaño que un **vector x** y contiene sumas acumuladas de los elementos del mismo. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, el vector resultante es $x = [1 \quad 6 \quad 9]$	`x=[1, 5, 3];`
		`cumsum(x)`
		`ans =`
		`1 6 9`

	Calcula una matriz que contiene la suma acumulada de los elementos en cada columna de una **matriz x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, la matriz resultante es $x = \begin{bmatrix} 1 & 5 & 3 \\ 3 & 9 & 9 \end{bmatrix}$	`x=[1, 5, 3; 2, 4, 6];`
		`cumsum(x)`
		`ans =`
		`1 5 3`
		`3 9 9`

`cumprod(x)`	Calcula un vector del mismo tamaño que un **vector x** y contiene productos acumulados de los elementos del mismo. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, el vector resultante es $x = [1 \quad 5 \quad 15]$	`x=[1, 5, 3];`
		`cumprod(x)`
		`ans =`
		`1 5 15`

	Calcula una matriz que contiene el producto acumulado de los elementos en cada columna de una **matriz x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, la matriz resultante es $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 20 & 18 \end{bmatrix}$	`x=[1, 5, 3; 2, 4, 6];`
		`cumprod(x)`
		`ans =`
		`1 5 3`
		`2 20 18`

3.5.4 Valores de ordenación

La tabla 3.8 menciona varios comandos para ordenar datos en una matriz, en orden ascendente o descendente.

3.5.5 Determinación del tamaño de matriz

MATLAB ofrece dos funciones (tabla 3.9) que le permiten determinar cuán grande es una matriz: **size** y **length**.

3.5.6 Varianza y desviación estándar

desviación estándar: medida de dispersión de los valores en un conjunto de datos

La desviación estándar y la varianza son medidas de cuánto varían los elementos de un conjunto de datos unos con respecto a otros. Todo estudiante sabe que la calificación promedio en un examen es importante, pero también es necesario conocer las calificaciones alta y baja para tener una idea de qué tan bien le fue. Las calificaciones de examen, como muchos tipos de

Tabla 3.8 Funciones de ordenamiento

`sort(x)`	Ordena los elementos de un vector **x** en orden ascendente. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, el vector resultante es $[1 \quad 3 \quad 5]$	`x=[1, 5, 3];`
		`sort(x)`
		`ans =`
		`  1  3  5`

	Ordena los elementos en cada columna de una matriz **x** en orden ascendente.
	Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, la matriz resultante es $x = \begin{bmatrix} 1 & 4 & 3 \\ 2 & 5 & 6 \end{bmatrix}$
		`x=[1, 5, 3; 2, 4, 6];`
		`sort(x)`
		`ans =`
		`  1  4  3`
		`  2  5  6`

`sort(x,'descend')`	Ordena los elementos en cada columna en orden descendente	`x=[1, 5, 3; 2, 4, 6];`
		`sort(x,'descend')`
		`ans =`
		`  2  5  6`
		`  1  4  3`

`sortrows(x)`	Ordena las filas en una matriz sobre la base de los valores en la primera columna y mantiene intacta cada fila.
	Por ejemplo, si $x = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 9 & 3 \\ 4 & 3 & 6 \end{bmatrix}$, entonces usar el comando **sortrows** moverá la fila media hacia la posición superior
		`x=[3,1,3;1,9,3;4,3,6]`
		`sortrows(x)`
		`ans =`
		`  1  9  3`
		`  3  1  2`
		`  4  3  6`

`sortrows(x,n)`	Ordena las filas en una matriz sobre la base de los valores en la columna **n**	`sortrows(x,2)`
		`ans =`
		`  3  1  2`
		`  4  3  6`
		`  1  9  3`

Tabla 3.9 Funciones de tamaño

`size(x)`	Determina el número de filas y columnas en la matriz **x**. (Si **x** es un arreglo multidimensional, **size** determina cuántas dimensiones existen y cuán grandes son.)	`x=[1, 5, 3; 2, 4, 6];`
		`size(x)`
		`ans =`
		`  2    3`

`[a,b] = size(x)`	Determina el número de filas y columnas en la matriz **x** y asigna el número de filas a **a** y el número de columnas a **b**	`[a,b]=size(x)`
		`a =`
		`  2`
		`b =`
		`  3`

`length(x)`	Determina la dimensión más grande de una matriz **x**	`x=[1, 5, 3; 2, 4, 6];`
		`length(x)`
		`ans =`
		`  3`

Ejercicio de práctica 3.7

Considere la siguiente matriz:

$$x = \begin{bmatrix} 4 & 90 & 85 & 75 \\ 2 & 55 & 65 & 75 \\ 3 & 78 & 82 & 79 \\ 1 & 84 & 92 & 93 \end{bmatrix}$$

1. Use la función **size** para determinar el número de filas y columna en esta matriz.
2. Use la función **sort** para ordenar cada columna en orden ascendente.
3. Use la función **sort** para ordenar cada columna en orden descendente.
4. Use la función **sortrows** para ordenar la matriz de modo que la primera columna esté en orden ascendente, pero cada fila todavía conserve sus datos originales. Su matriz se debe parecer a ésta:

$$x = \begin{bmatrix} 1 & 84 & 92 & 93 \\ 2 & 55 & 65 & 75 \\ 3 & 78 & 82 & 79 \\ 4 & 90 & 85 & 75 \end{bmatrix}$$

EJEMPLO 3.3

Datos del clima

El National Weather Service recopila cantidades masivas de datos del clima todos los días (figura 3.5). Dichos datos están disponibles a todas las personas en el servicio en línea de la agencia en http://cdo.ncdc.noaa.gov/CDO/cdo. El análisis de grandes cantidades de datos puede ser confuso, así que es buena idea comenzar con un pequeño conjunto de datos, desarrollar un enfoque que funcione y luego aplicarlo a conjuntos de datos más grandes en los que se esté interesado.

Del National Weather Service se extrajo información de precipitaciones para una localidad para todo 1999 y se almacenó en un archivo llamado Weather_Data.xls. (La .xls indica que los datos están en una hoja de cálculo de Excel.) Cada fila representa un mes, de modo que hay 12 filas, y cada columna representa el día del mes (1 a 31), de modo que hay 31 columnas. Dado que no todos los meses tienen el mismo número de días, existen datos perdidos para

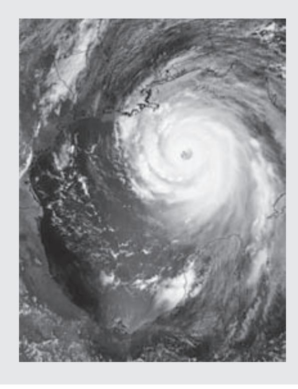


Figura 3.5
Fotografía satelital de un huracán. (Cortesía de NASA/Jet Propulsion Laboratory.)

Tabla 3.10 Datos de precipitación de Asheville, Carolina del Norte

1999	Día 1	Día 2	Día 3	Día 4		Día 28	Día 29	Día 30	Día 31
Enero	0	0	272	0	etc. . . .	0	0	33	33
Febrero	61	103	0	2		62	−99999	−99999	−99999
Marzo	2	0	17	27		0	5	8	0
Abril	260	1	0	0		13	86	0	−99999
Mayo	47	0	0	0		0	0	0	0
Junio	0	0	30	42		14	14	8	−99999
Julio	0	0	0	0		5	0	0	0
Agosto	0	45	0	0		0	0	0	0
Septiembre	0	0	0	0		138	58	10	−99999
Octubre	0	0	0	14		0	0	0	1
Noviembre	1	163	5	0		0	0	0	−99999
Diciembre	0	0	0	0		0	0	0	0

algunas localidades en muchas de las últimas columnas. Se indica que los datos están perdidos para dichas localidades al colocar en ellos el número −99999. La información de precipitación se presenta en centésimas de pulgada. Por ejemplo, el 1 de febrero hubo 0.61 pulgadas de precipitación, y el 1 de abril hubo 2.60 pulgadas de precipitación. En la tabla 3.10 se presenta una muestra de los datos, con etiquetas para dar claridad; sin embargo, **los datos en el archivo sólo contienen números**.

Use los datos en el archivo para encontrar lo siguiente:

a. la precipitación total en cada mes.
b. la precipitación total durante el año.
c. el mes y día en que se registró la precipitación máxima durante el año.

1. Establezca el problema.
   Con los datos del archivo Weather_Data.xls, encontrar la precipitación total mensual, la precipitación total del año y el día en el que llovió más.
2. Describa las entradas y salidas.

   Entrada La entrada para este ejemplo se incluye en un archivo de datos llamado Weather_Data.xls y consiste en una matriz bidimensional. Las filas representan un mes y las columnas representan un día.

   Salida La salida debe ser la precipitación total para cada mes, la precipitación total para el año y el día en el que la precipitación fue un máximo. Se decidió presentar la precipitación en pulgadas pues en el enunciado del problema no se especificaron otras unidades.
3. Desarrolle un ejemplo a mano.
   Para el ejemplo a mano, trate sólo con un pequeño subconjunto de los datos. La información que se incluye en la tabla 3.10 es suficiente. El total para enero, días 1 a 4, es

$$total_1 = (0 + 0 + 272 + 0)/100 = 2.72 \text{ pulgadas}$$

El total para febrero, días 1 a 4, es

total_2 = (61 + 103 + 0 + 2)/100 = 1.66 pulgadas

Ahora sume los meses para obtener el total combinado. Si la muestra "año" sólo es enero y febrero, entonces

total = total_1 + total_2 = 2.72 + 1.66 = 4.38 pulgadas

Para encontrar el día en el que ocurrió la precipitación máxima, primero encuentre el máximo en la tabla y luego determine cuál fila y cuál columna está en ella.

El trabajar en un ejemplo a mano le permite formular los pasos que se requieren para resolver el problema en MATLAB.

4. Desarrolle una solución MATLAB.
Primero se necesitará guardar el archivo de datos en MATLAB como una matriz. Puesto que el archivo es una hoja de cálculo de Excel, la manera más sencilla es usar el Import Wizard (asistente de importaciones). Haga doble clic en el archivo en la ventana de directorio actual para lanzar el Import Wizard.

Una vez que el Import Wizard complete la ejecución, en la ventana del área de trabajo aparecerá el nombre de variable **Sheet1**. (Véase la figura 3.6; su versión puede nombrar la variable **Weather_data**.)

Puesto que no todos los meses tienen 31 días, existen algunas entradas para días no existentes. En dichos campos se insertó el valor −99999. Puede hacer doble clic en el nombre de variable, **Sheet1**, en la ventana del área de trabajo, para editar esta matriz y cambiar los valores "fantasma" a 0. (Véase la figura 3.7.)

Ahora escriba el archivo-m script para resolver el problema:

```
clc
%Ejemplo 3.3-Datos del clima
%En este ejemplo se encontrará la precipitación total
%para cada mes, y para todo el año, con un archivo de datos
```

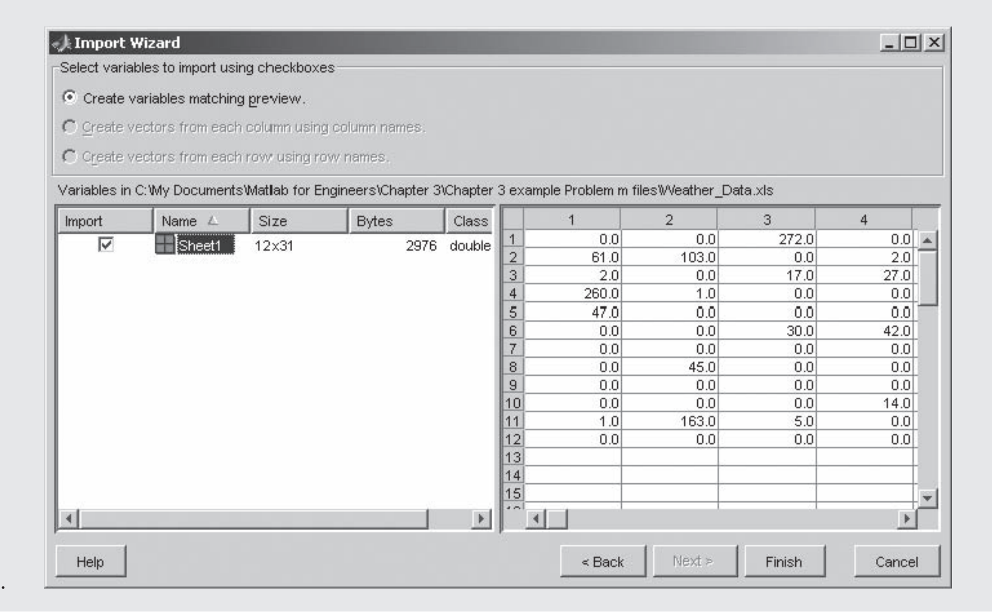


Figura 3.6
Import Wizard de MATLAB.

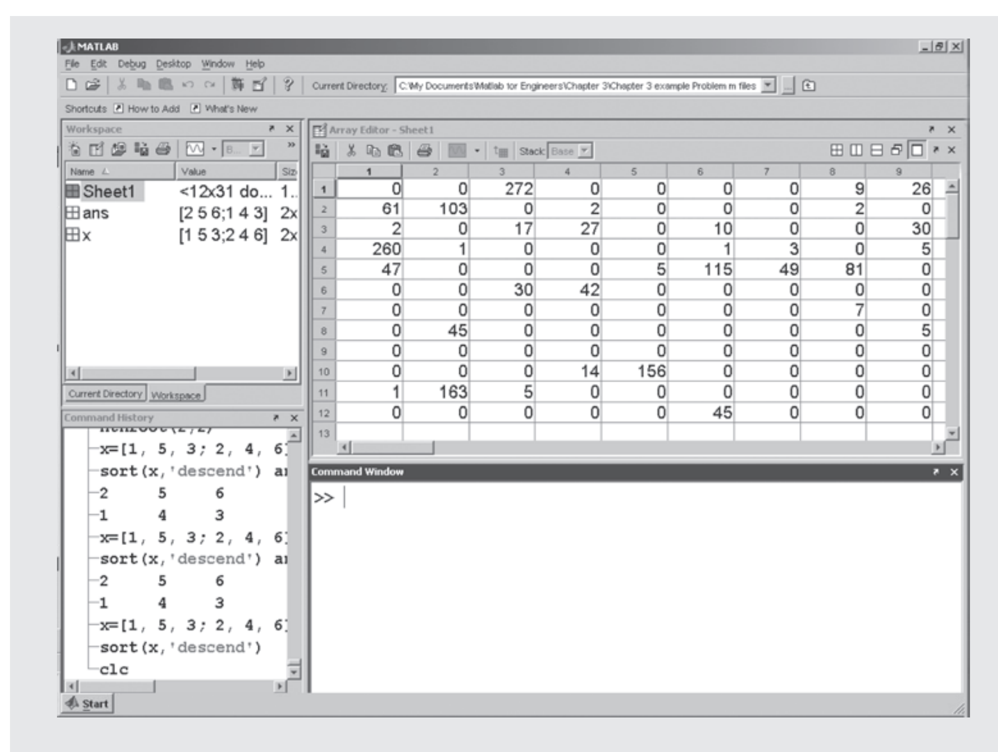


Figura 3.7
Editor de arreglos de MATLAB. En esta ventana se puede editar el arreglo y cambiar todos los "valores fantasma" de −99999 a 0.

```
%También se encontrarán el mes y día cuando la
%precipitación fue la máxima
%
wd=Sheet1;
%Use el operador transpuesto para cambiar filas a columnas
wd = wd';
%Encuentra la suma de cada columna, que es la suma para cada
%mes
monthly_total=sum(wd)/100
%Encuentra el total anual
yearly_total = sum(monthly_total)
%Encuentra el máximo anual y el día cuando ocurre
[maximum_precip,month]=max(max(wd))
%Encuentra el máximo anual y el mes cuando ocurre
[maximum_precip,day]=max(max(wd'))
```

Note que el código no comienza con los usuales comandos **clear, clc**, porque ello limpiaría el área de trabajo, lo que efectivamente borraría la variable **Sheet1**. A continuación se renombra **Sheet1** a **wd**, pues es más corto de escribir (y representa weather_data: datos del clima). Puesto que la variable se usará mucho, es buena idea hacerla corta, para minimizar las posibilidades de errores causadas por errores de escritura.

A continuación, se traspone la matriz **wd**, de modo que los datos para cada mes están en una columna en lugar de una fila. Esto permite el uso del comando **sum** para sumar todos los valores de precipitación para el mes.

Ahora se pueden sumar todos los totales mensuales para obtener el total para el año. Una sintaxis alternativa es

```
yearly_total = sum(sum(wd))
```

Encontrar la precipitación máxima diaria es sencillo; lo que hace difícil este ejemplo es determinar el día y mes cuando ocurrió el máximo. El comando

```
[maximum_precip,month] = max(max(wd))
```

es más fácil de entender si se descompone en dos comandos. Primero,

```
[a,b] = max(wd)
```

regresa una matriz de máximos para cada columna, que en este caso es el máximo para cada mes. Este valor se asigna al nombre de variable **a**. La variable **b** se convierte en una matriz de números índice que representan la fila en cada columna en la que ocurrió el máximo. Entonces, el resultado es

```
a =
      Columns 1 through 9
       272   135    78   260   115   240   157   158   138
      Columns 10 through 12
       156   255    97
b =
      Columns 1 through 9
         3    18    27     1     6    25    12    24    28
      Columns 10 through 12
         5    26    14
```

Ahora, cuando se ejecuta el comando **max** la segunda vez, se determina la precipitación máxima para todo el conjunto de datos, que es el valor máximo en la matriz **a**. Además, a partir de la matriz **a**, se encuentra el número índice para dicho máximo:

```
[c,d]=max(a)
c =
       272
d =
       1
```

Estos resultados dicen que la precipitación máxima ocurrió en la columna 1 de la matriz **a**, lo que significa que ocurrió en el primer mes.

De igual modo, la trasposición de la matriz **wd** (es decir, obtener **wd'**) y encontrar el máximo dos veces permite encontrar el día del mes en el que ocurrió el máximo.

Existen varias cosas que debe notar acerca de la pantalla MATLAB que se muestra en la figura 3.8. En la **ventana del área de trabajo**, se mencionan tanto **Sheet1** como **wd. Sheet1** es una matriz 12 × 31, mientras que **wd** es una matriz 31 × 12. Todas las variables que se crearon cuando se ejecutó el archivo-m ahora están disponibles a la ventana de comandos. Esto hace fácil la realización de cálculos adicionales en la ventana de comandos después de que el archivo-m completa su corrida. Por ejemplo, note que se olvidó cambiar el valor **maximum_precip** de centésimas de pulgada a pulgadas. Agregar el comando

```
maximum_precip=maximum_precip/100
```

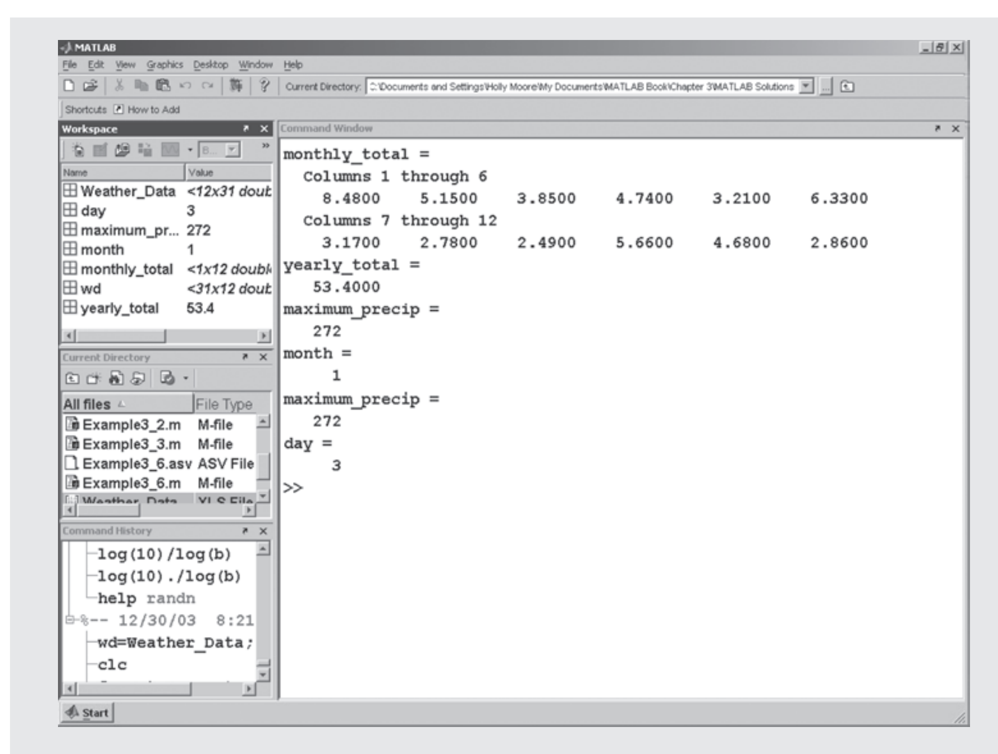


Figura 3.8
Resultados de los cálculos de precipitación.

corregiría dicho error. Note también que el archivo Weather_Data.xls todavía está en el directorio actual. Finalmente, note que la **ventana de historia de comandos** refleja sólo los comandos emitidos desde la **ventana de comandos**; no muestra los comandos ejecutados desde un archivo-m.

5. Ponga a prueba la solución.
   Abra el archivo Weather_Data.xls y confirme que la precipitación máxima ocurrió el 3 de enero. Una vez que confirme que su programa archivo-m funciona, puede usarlo para analizar otros datos. El National Weather Service mantiene registros similares para todas sus estaciones de registro.

datos que son importantes en ingeniería, con frecuencia se distribuyen en una curva con forma de "campana". En una distribución normal (gaussiana) de una gran cantidad de datos, aproximadamente el 68% de los datos cae dentro de una desviación estándar (sigma) de la media (± un sigma). Si extiende el rango a una variación de dos sigma (± dos sigma), aproximadamente 95% de los datos deben caer dentro de estas fronteras, y si va hacia las tres sigma, más del 99% de los datos deben caer en este rango (figura 3.9). Por lo general, mediciones como la desviación estándar y la varianza son significativas sólo con grandes conjuntos de datos.

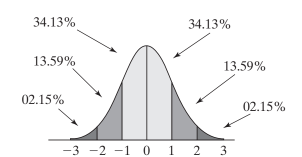


Figura 3.9
Distribución normal.

varianza: desviación estándar al cuadrado

Considere los datos que se grafican en la figura 3.10. Ambos conjuntos de datos tienen el mismo valor promedio (media) de 50. Sin embargo, es fácil ver que el primer conjunto de datos tiene más variación que el segundo.

La definición matemática de varianza es

$$\text{varianza} = \sigma^2 = \frac{\sum_{k=1}^{N}(x_k - \mu)^2}{N - 1}$$

En esta ecuación, el símbolo μ representa la media de los valores x_k en el conjunto de datos. Por ende, el término $x_k - \mu$ simplemente es la diferencia entre el valor real y el valor promedio. Los términos se elevan al cuadrado y se suman:

$$\sum_{k=1}^{N}(x_k - \mu)^2$$

Finalmente, se divide el término suma entre el número de valores en el conjunto de datos (N), menos 1.

La desviación estándar (σ), que se usa con más frecuencia que la varianza, sólo es la raíz cuadrada de la varianza.

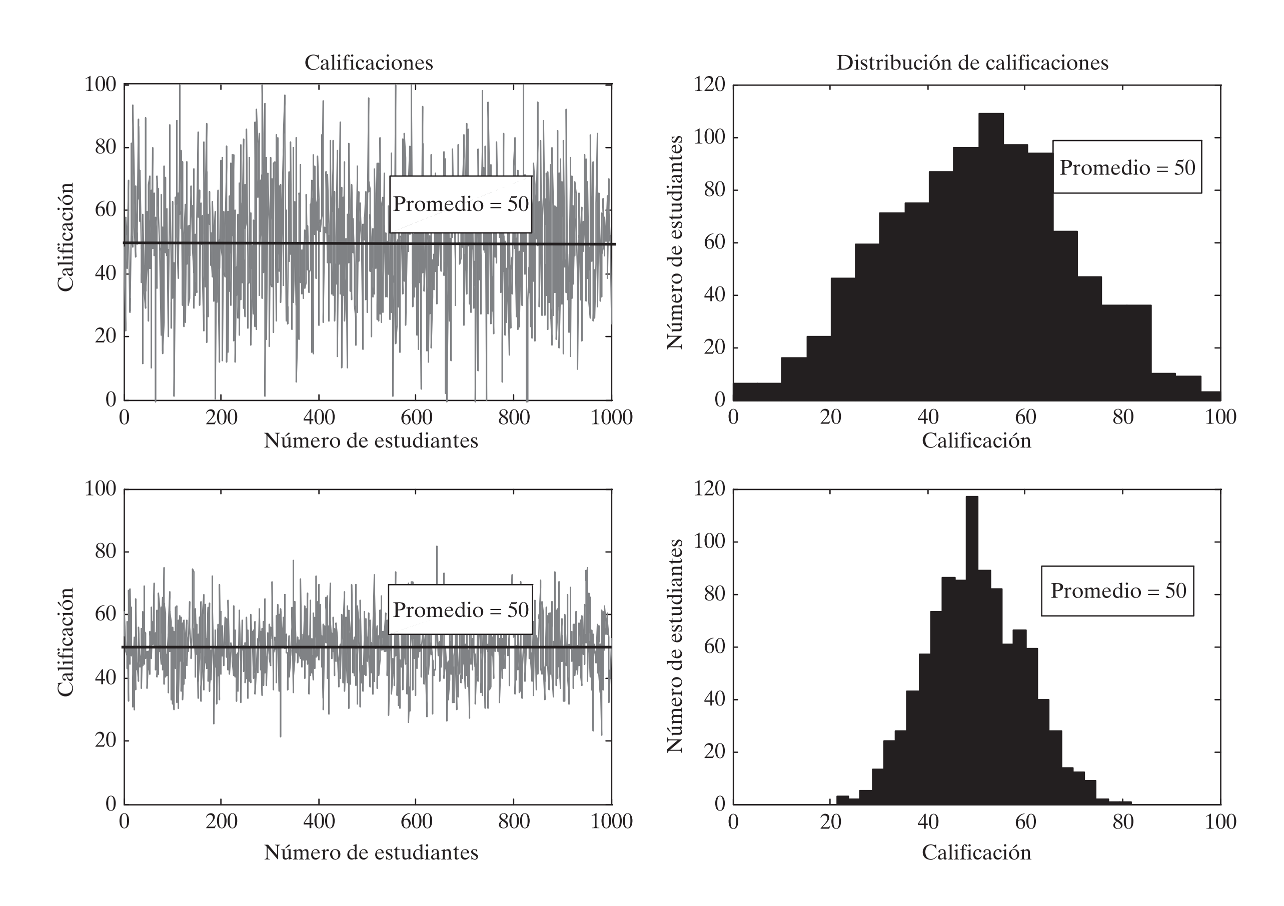


Figura 3.10
Calificaciones para dos pruebas diferentes.

La función MATLAB que se usa para encontrar la desviación estándar es **std**. Cuando esta función se aplica en el gran conjunto de datos que se muestra en la figura 3.10, se obtiene la siguiente salida:

```
std(scores1)
ans =
    20.3653
std(scores2)
ans =
    9.8753
```

En otras palabras, aproximadamente 68% de los datos en el primer conjunto de datos cae entre el promedio, 50, y ±20.3653. De igual modo, 68% de los datos en el segundo conjunto de datos cae entre el mismo promedio, 50, y ±9.8753.

La varianza se encuentra en forma similar con la función **var**:

```
var(scores1)
ans =
   414.7454
var(scores2)
ans =
 97.5209
```

En la tabla 3.11 se muestra la sintaxis para calcular tanto la desviación estándar como la varianza.

Tabla 3.11 Funciones estadísticas

`std(x)`	Calcula la desviación estándar de los valores en un vector **x**. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, la desviación estándar es 2. Sin embargo, para muestras pequeñas de datos, usualmente no se calculan desviaciones estándar	`x=[1, 5, 3];`
		`std(x)`
		`ans =`
		`  2`

	Regresa un vector fila que contiene la desviación estándar calculada para cada columna de una matriz **x**. Por ejemplo, si $x = \begin{bmatrix} 1 & 5 & 3 \\ 2 & 4 & 6 \end{bmatrix}$, la desviación estándar en la columna 1 es 0.7071, la desviación estándar en la columna 2 es 0.7071 y la desviación estándar en la columna 3 es 2.1213
	De nuevo, para muestras pequeñas de datos, usualmente no se calculan desviaciones estándar
		`x=[1, 5, 3; 2, 4, 6];`
		`std(x)`
		`ans =`
		`0.7071    0.7071`
		`2.1213`

`var(x)`	Calcula la varianza de los datos en **x**. Por ejemplo, si $x = [1 \quad 5 \quad 3]$, la varianza es 4. Sin embargo, para muestras pequeñas de datos, usualmente no se calcula la varianza. Note que la desviación estándar en este ejemplo es la raíz cuadrada de la varianza	`var(x)`
		`ans =`
		`  4`

Ejercicio de práctica 3.8

Considere la siguiente matriz:

$$x = \begin{bmatrix} 4 & 90 & 85 & 75 \\ 2 & 55 & 65 & 75 \\ 3 & 78 & 82 & 79 \\ 1 & 84 & 92 & 93 \end{bmatrix}$$

1. Encuentre la desviación estándar para cada columna.
2. Encuentre la varianza para cada columna.
3. Calcule la raíz cuadrada de la varianza que encontró para cada columna.
4. ¿Cómo se comparan los resultados del problema 3 contra la desviación estándar que encontró en el problema 1?

EJEMPLO 3.4

Datos climatológicos

Los climatólogos examinan datos del clima durante largos periodos de tiempo, con la intención de encontrar un patrón. En Estados Unidos, desde 1850, se conservan datos confiables del clima; sin embargo, la mayoría de las estaciones de registro sólo se asentaron desde 1930 y 1940 (figura 3.11). Los climatólogos realizan cálculos estadísticos sobre los datos que recopilan. Aunque los datos en Weather_Data.xls representan sólo una localidad para un año, se pueden usar los datos para practicar cálculos estadísticos. Encuentre la precipitación diaria media para cada mes y la precipitación diaria media para el año, y luego encuentre la desviación estándar para cada mes y para el año.

1. Establezca el problema.
   Encontrar la precipitación diaria media para cada mes y para el año, sobre la base de los datos en Weather_Data.xls. Además, encontrar la desviación estándar de los datos durante cada mes y durante todo el año.
2. Describa las entradas y salidas.

   Entrada Use el archivo Weather_Data.xls como entrada al problema

   Salida Encontrar

   la precipitación diaria media para cada mes.
   la precipitación diaria media para el año.

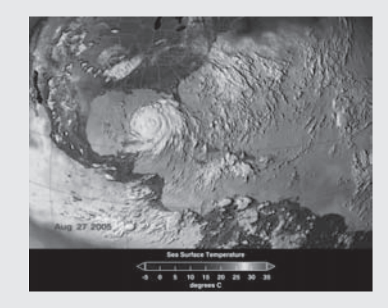


Figura 3.11
Un huracán sobre Florida. (Cortesía de NASA/Jet Propulsion Laboratory.)

la desviación estándar de los datos de precipitación diaria para cada mes.
la desviación estándar de los datos de precipitación diaria para el año.

3. Desarrolle un ejemplo a mano.
Use sólo los datos para los primeros cuatro días del mes:
Promedio de enero = (0 + 0 + 272 + 0)/4 = 68 centésimas de pulgada de precipitación, o 0.68 de pulgada.
La desviación estándar se encuentra de la siguiente ecuación:

$$\sigma = \sqrt{\frac{\sum_{k=1}^{N}(x_k - \mu)^2}{N - 1}}$$

Use sólo los primeros cuatro días de enero, calcule primero la suma de los cuadrados de la diferencia entre la media y el valor real:

$$(0 - 68)^2 + (0 - 68)^2 + (272 - 68)^2 + (0 - 68)^2 = 55{,}488$$

Divida entre el número de puntos de datos menos 1:

$$55{,}488/(4 - 1) = 18{,}496$$

Finalmente, saque la raíz cuadrada, para obtener 136 centésimas de pulgada de precipitación, o 1.36 pulgadas.

4. Desarrolle una solución MATLAB.
Primero se necesita cargar el archivo Weather_Data.xls y edite las entradas −99999. Aunque esto se podría hacer en una forma similar al proceso descrito en el ejemplo 3.3, existe una forma más sencilla: se podrían guardar los datos del ejemplo 3.3 en un archivo, de modo que estén disponibles para usar más adelante. Si se quiere guardar toda el área de trabajo, sólo escriba

```
save <filename>
```

donde **filename** es un nombre de archivo definido por el usuario. Si sólo quiere guardar una variable, escriba

```
save <filename>  <variable_name>
```

que guarda una sola variable o una lista de variables a un archivo. Todo lo que se requiere guardar es la variable **wd**, de modo que el siguiente comando es suficiente:

```
save wd wd
```

Este comando guarda la matriz **wd** en el archivo **wd.mat**. Verifique la ventana de directorio actual para asegurarse de que se almacenó **wd.mat** (figura 3.12).
Ahora, el archivo-m que se creó para resolver este ejemplo puede cargar los datos automáticamente:

```
clear, clc
%  Ejemplo 3.4 Datos climatológicos
%  En este ejemplo, se encuentra la precipitación diaria
%  media para cada mes
%  y la precipitación diaria media par el año
%  También se encuentra la desviación estándar de los datos
%
```

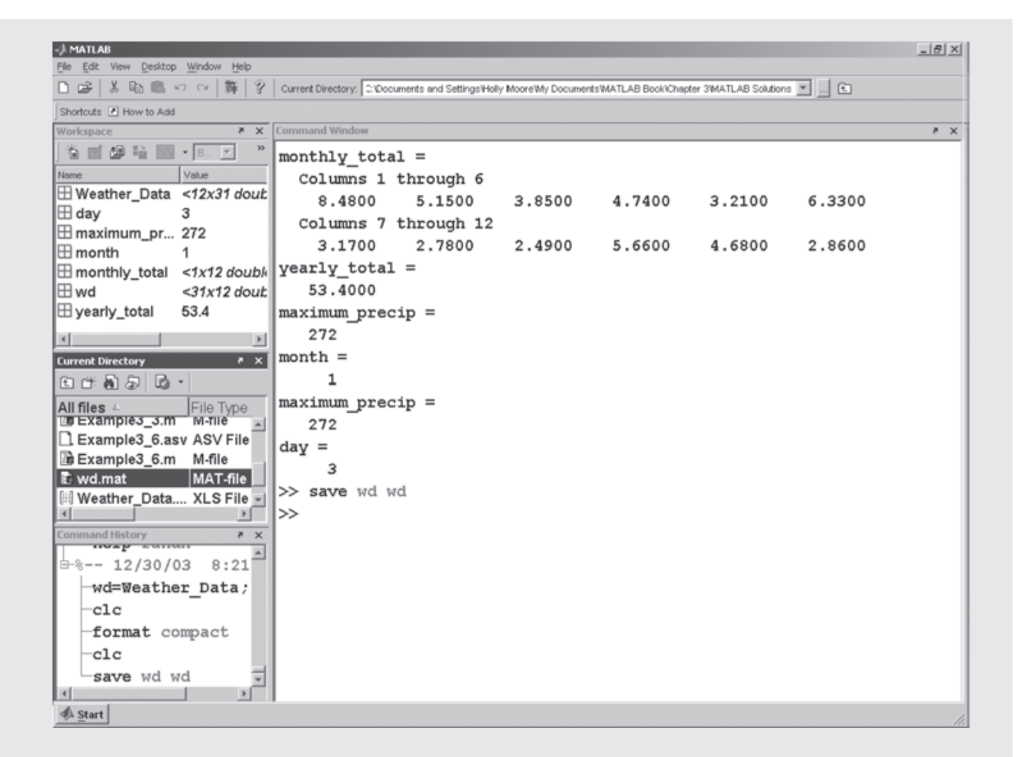


Figura 3.12
El directorio actual registra el nombre del archivo guardado.

```
%  Cambiar el formato a bank con frecuencia hace la salida
%  más fácil de leer
format bank
%  Al guardar la variable wd del último ejemplo, está
%  disponible para usar en este ejemplo
load wd
Average_daily_precip_monthly = mean(wd)
Average_daily_precip_yearly = mean(wd(:))
%  Otra forma de encontrar la precipitación anual promedio
Average_daily_precip_yearly = mean(mean(wd))
% Ahora calcula la desviación estándar
Monthly_Stdeviation = std(wd)
Yearly_Stdeviation = std(wd(:))
```

Los resultados, que se muestran en la ventana de comando, son

```
Average_daily_precip_monthly =
  Columns 1 through 3
     27.35     16.61     12.42
  Columns 4 through 6
     15.29     10.35     20.42
  Columns 7 through 9
     10.23     8.97     8.03
  Columns 10 through 12
     18.26     15.10     9.23
```

```
Average_daily_precip_yearly =
     14.35
Average_daily_precip_yearly =
     14.35
Monthly_Stdeviation =
  Columns 1 through 3
     63.78     35.06     20.40
  Columns 4 through 6
     48.98     26.65     50.46
  Columns 7 through 9
     30.63     30.77     27.03
  Columns 10 through 12
     42.08     53.34     21.01
Yearly_Stdeviation =
     39.62
```

La precipitación diaria media para el año se calculó en dos formas equivalentes. Se encontró la media de cada mes y luego la media (promedio) de los valores mensuales. Esto resulta ser igual que tomar la media de todos los datos a la vez. En este ejemplo se introdujo una nueva sintaxis. El comando

```
wd(:)
```

convierte la matriz bidimensional **wd** en una matriz unidimensional, lo que hace posible encontrar la media en un paso.

La situación es diferente para la desviación estándar de la precipitación diaria para el año. Aquí, es necesario realizar sólo un cálculo:

```
std(wd(:))
```

De otro modo encontraría la desviación estándar de la desviación estándar, no lo que quiere en absoluto.

5. Ponga a prueba la solución.
Primero, compruebe los resultados para asegurarse de que tienen sentido. Por ejemplo, la primera vez que se ejecutó el archivo-m, la matriz **wd** todavía contenía valores −99999. Esto resultó en valores media menores que 1. Puesto que no es posible tener lluvia negativa, la comprobación de la razonabilidad de los datos alertó el problema. Finalmente, aunque calcular la lluvia diaria media para un mes a mano serviría como una comprobación excelente, sería tedioso. Puede usar MATLAB para ayudarse a calcular la media sin usar una función predefinida. La ventana de comandos es un lugar conveniente para realizar dichos cálculos:

```
load wd
sum(wd(:,1))    %Encuentra la suma de todas las filas en la
                 columna uno
%de la matriz wd
ans =
    848.00
ans/31
ans =
    27.35
```

Compare estos resultados con los de enero (mes 1).

Sugerencia

Use el operador dos puntos para cambiar una matriz bidimensional en una sola columna:

A = X(:)

3.6 NÚMEROS ALEATORIOS

Con frecuencia, en los cálculos de ingeniería se usan números aleatorios como parte de una simulación de datos medidos. Los datos medidos rara vez se comportan exactamente como predicen los modelos matemáticos, así que se pueden agregar pequeños valores de números aleatorios a las predicciones para hacer que un modelo se comporte más como un sistema real. Los números aleatorios también se usan para modelar juegos de azar. En MATLAB se pueden generar dos diferentes tipos de números aleatorios: números aleatorios uniformes y números aleatorios gaussianos (con frecuencia llamados una distribución normal).

3.6.1 Números aleatorios uniformes

Los números aleatorios uniformes se generan con la función **rand**. Estos números se distribuyen de forma pareja entre 0 y 1. (Consulte la función help para más detalles.) La tabla 3.12 cita varios comandos MATLAB para generar números aleatorios.

Se puede crear un conjunto de números aleatorios sobre otros rangos al modificar los números creados por la función **rand**. Por ejemplo, para crear un conjunto de 100 números distribuidos de manera pareja entre 0 y 5, primero cree un conjunto sobre el rango por defecto con el comando

```
r = rand(100,1);
```

Ahora sólo necesita multiplicar por 5 para expandir el rango a 0-5:

```
r = r * 5;
```

Tabla 3.12 Generadores de números aleatorios

`rand(n)`	Regresa una matriz $n \times n$. Cada valor en la matriz es un número aleatorio entre 0 y 1	`rand(2)`
		`ans =`
		`0.9501 0.6068`
		`0.2311 0.4860`

`rand(m,n)`	Regresa una matriz $m \times n$. Cada valor en la matriz es un número aleatorio entre 0 y 1	`rand(3,2)`
		`ans =`
		`0.8913 0.0185`
		`0.7621 0.8214`
		`0.4565 0.4447`

`randn(n)`	Regresa una matriz $n \times n$. Cada valor en la matriz es un número aleatorio gaussiano (o normal) con una media de 0 y una varianza de 1	`randn(2)`
		`ans =`
		`-0.4326 0.1253`
		`-1.6656 0.2877`

`randn(m,n)`	Regresa una matriz $m \times n$. Cada valor en la matriz es un número aleatorio gaussiano (o normal) con una media de 0 y una varianza de 1	`randn(3,2)`
		`ans =`
		`-1.1465 -0.0376`
		`1.1909 0.3273`
		`1.1892 0.1746`

Si se quiere cambiar el rango a 5-10, se puede sumar 5 a cada valor en el arreglo:

```
r = r+5;
```

El resultado serán números aleatorios que varíen de 5 a 10. Estos resultados se pueden generalizar con la ecuación

$$x = (\text{máx} - \text{mín}) \cdot \text{conjunto_números_aleatorios} + \text{media}$$

3.6.2 Números aleatorios gaussianos

Los números aleatorios gaussianos tienen la distribución normal que se muestra en la figura 3.9. No hay límite absoluto superior o inferior a un conjunto de datos de este tipo; sólo se vuelve cada vez menos probable encontrar datos más alejados de la media que se tiene. Los conjuntos de números aleatorios gaussianos se describen al especificar su promedio y la desviación estándar del conjunto de datos.

MATLAB genera valores gaussianos con una media de 0 y una varianza de 1.0, con la función **randn**. Por ejemplo,

```
randn(3)
```

regresa

```
ans =
 -0.4326    0.2877    1.1892
 -1.6656   -1.1465   -0.0376
  0.1253    1.1909    0.3273
```

Si se necesita un conjunto de datos con un promedio diferente o una desviación estándar diferente, se comienza con el conjunto por defecto de números aleatorios y luego se modifica. Dado que la desviación estándar por defecto es 1, se debe *multiplicar* por la desviación estándar requerida para el nuevo conjunto de datos. Puesto que la media por defecto es 0, se necesitará *sumar* la nueva media:

$$x = \text{desviación_estándar} \cdot \text{conjunto_datos_aleatorios} + \text{media}$$

Por ejemplo, para crear una secuencia de 500 variables aleatorias gaussianas con una desviación estándar de 2.5 y una media de 3, escriba

```
x = randn(1,500)*2.5 + 3;
```

Ejercicio de práctica 3.9

1. Cree una matriz 3 × 3 de números aleatorios distribuidos de manera pareja.
2. Cree una matriz 3 × 3 de números aleatorios distribuidos de manera normal.
3. Cree una matriz 100 × 5 de números aleatorios distribuidos de manera pareja. Asegúrese de suprimir la salida.
4. Encuentre el máximo, la desviación estándar, la varianza y la media para cada columna en la matriz que creó en el problema 3.
5. Cree una matriz 100 × 5 de números aleatorios distribuidos de manera normal. Asegúrese de suprimir la salida.
6. Encuentre el máximo, la desviación estándar, la varianza y la media para cada columna en la matriz que creó en el problema 5.
7. Explique por qué son diferentes sus resultados para los problemas 4 y 6.

EJEMPLO 3.5

Ruido

Los números aleatorios se pueden usar para simular el ruido que se escucha como estática en el radio. Al agregar este ruido a los archivos de datos que almacenan música, se puede estudiar el efecto de la estática en las grabaciones.

MATLAB tiene la habilidad de tocar archivos de música mediante la función **sound**. Para demostrar esta función, también tiene un archivo de música interno con un corto segmento del *Mesías* de Handel. En este ejemplo, se usará la función **randn** para crear ruido y luego se agregará el ruido al clip de música.

La música se almacena en MATLAB como un arreglo con valores desde −1 hasta 1. Para convertir este arreglo en música, la función **sound** requiere una frecuencia muestra. El archivo **handel.mat** contiene tanto un arreglo que representa la música como el valor de la frecuencia muestra. Para escuchar el *Mesías*, primero debe cargar el archivo, con el comando

```
load handel
```

Note que dos nuevas variables (**y** y **Fs**) se agregan a la ventana del área de trabajo cuando se carga el archivo **handel**. Para tocar el clip, escriba

```
sound(y, Fs)
```

Experimente con diferentes valores de **Fs** para escuchar el efecto de diferentes frecuencias muestra sobre la música. Claramente, el sonido se debe meter en su computadora o no será capaz de escuchar la música.

1. Establezca el problema.
   Añadir un componente de ruido a la grabación del *Mesías* de Handel, que se incluye con MATLAB.
2. Describa las entradas y salidas.
   Entrada Archivo de datos MATLAB del *Mesías* de Handel, almacenado como el archivo interno **handel**
   Salida Un arreglo que representa el *Mesías*, con estática añadida
   Una gráfica de los primeros 200 elementos del archivo de datos
3. Desarrolle un ejemplo a mano.
   Dado que los datos en el archivo de música varían entre −1 y +1, se deben agregar valores de ruido de un orden de magnitud más pequeño. Primero se intentarán valores centrados en 0 y con una desviación estándar de 0.1.
4. Desarrolle una solución MATLAB.

```
%Example 3.5
%Noise
load handel      %Carga el archivo de datos de música
sound(y,Fs)      %Toca el archivo de datos de música
pause            %Pausa para escuchar la música
```



Figura 3.13
Orquesta Sinfónica de Utah.

```
% Asegúrese de dar enter para continuar después de tocar la música
% Agrega ruido aleatorio
noise=randn(length(y),1)*0.10;
sound(y+noise,Fs)
```

Este programa le permite tocar la grabación del *Mesías* tanto con el ruido agregado como sin él. Puede ajustar el multiplicador en la línea de ruido para observar el efecto de cambiar la magnitud de la estática agregada. Por ejemplo:

```
noise=randn(length(y),1)*0.20
```

5. Ponga a prueba la solución.
   Además de tocar de nuevo la música con y sin ruido agregado, podría graficar los resultados. Puesto que el archivo es bastante grande (73,113 elementos), sólo se graficarán los primeros 200 puntos:

```
%  Grafica los primeros 200 puntos de datos en cada archivo
t=1:length(y);
plot(t(1,1:200),y(1:200,1),t(1,1:200),noise(1:200,1),':')
title('Mesías de Handel')
xlabel('Número de elementos en el arreglo de música')
ylabel('Frecuencia')
```

Estos comandos le dicen a MATLAB que grafique el número índice de los datos en el eje x y el valor almacenado en los arreglos de música en el eje y.

En la figura 3.14, la línea sólida representa los datos originales y la línea punteada son los datos a los que se agregó ruido. Como se esperaba, los datos ruidosos tienen un rango mayor y no siempre siguen el mismo patrón que el original.

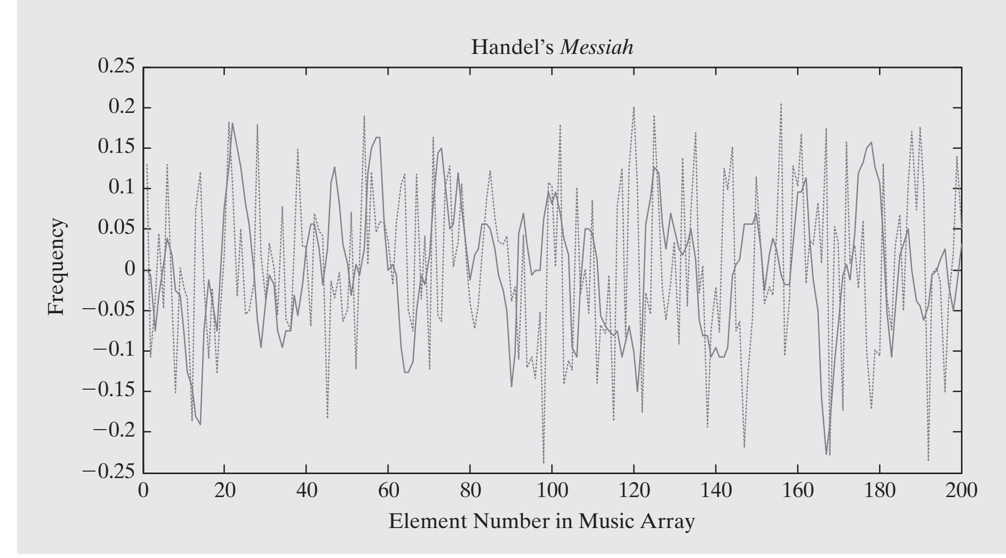


Figura 3.14
Mesías de Handel.

3.7 NÚMEROS COMPLEJOS

MATLAB incluye varias funciones que se usan principalmente con números complejos. Los números complejos consisten de dos partes: una parte real y un componente imaginario. Por ejemplo,

$$5 + 3i$$

número complejo: número con componentes real e imaginario

es un número complejo. El componente real es 5 y el componente imaginario es 3. Los números complejos se pueden ingresar en MATLAB de dos formas: como un problema de suma, como

```
A=5 + 3i            o            A=5+3*i
```

o con la función **complex**, como en

```
A=complex(5,3)
```

que regresa

```
A =
  5.0000 + 3.0000i
```

Como es estándar en MATLAB, la entrada a la función **complex** puede ser dos escalares o dos arreglos de valores. Por ende, si **x** y **y** se definen como

```
x=1:3;
y=[-1,5,12];
```

entonces se puede usar la función **complex** para definir un arreglo de números complejos del modo siguiente:

```
complex(x,y)
ans =
  1.0000 - 1.0000i   2.0000 + 5.0000i   3.0000 +12.0000i
```

Se pueden usar las funciones **real** e **imag** para separar los componentes real e imaginario de los números complejos. Por ejemplo, para **A=5 + 3*i**, se tiene

```
real(A)
ans =
  5
imag(A)
ans =
  3
```

La función **isreal** se puede usar para determinar si una variable almacena un número complejo. Regresa 1 si la variable es real y 0 si es compleja. Dado que **A** es un número complejo, se obtiene

```
isreal(A)
ans =
  0
```

Por tanto, la función **isreal** es falsa y regresa un valor de 0.

La conjugada compleja de un número complejo consiste en el mismo componente real, pero un componente imaginario de signo opuesto. La función **conj** regresa la conjugada compleja:

```
conj(A)
ans =
  5.0000 - 3.0000i
```

El operador transpuesto también regresa la conjugada compleja de un arreglo, además de convertir filas a columnas y columnas a filas. Por tanto, se tiene

```
A'
ans =
  5.0000 - 3.0000i
```

Desde luego, en este ejemplo **A** es un escalar. Se puede crear un arreglo complejo **B** con el uso de **A** y realizar operaciones de suma y multiplicación:

```
B=[A, A+1, A*3]
B =
  5.0000 + 3.0000i   6.0000 + 3.0000i   15.0000 + 9.0000i
```

El transpuesto de **B** es

```
B'
ans =
   5.0000 - 3.0000i
   6.0000 - 3.0000i
  15.0000 - 9.0000i
```

Con frecuencia, los números complejos se consideran como la descripción de una posición en el plano *x-y*. La parte real del número corresponde al valor *x*, y el componente imaginario corresponde al valor *y*, como se muestra en la figura 3.15a. Otra forma de pensar acerca de este punto es describirlo con coordenadas polares; esto es: con un radio y un ángulo (figura 3.15b).

MATLAB incluye funciones para convertir números complejos de forma cartesiana a polar.

Cuando la función valor absoluto se usa con un número complejo, calcula el radio mediante el teorema de Pitágoras:

coordenadas polares: técnica para describir una ubicación con el uso de un ángulo y una distancia

```
abs(A)
ans =
    5.8310
```

$$\text{radio} = \sqrt{(\text{componente real})^2 + (\text{componente imaginario})^2}$$

Puesto que, en este ejemplo, el componente real es 5 y el componente imaginario es 3,

$$\text{radio} = \sqrt{5^2 + 3^2} = 5.8310$$

También se podría calcular el radio en MATLAB con el uso de las funciones **real** e **imag** descritas anteriormente:

```
sqrt(real(A).^2 + imag(A).^2)
ans =
    5.8310
```

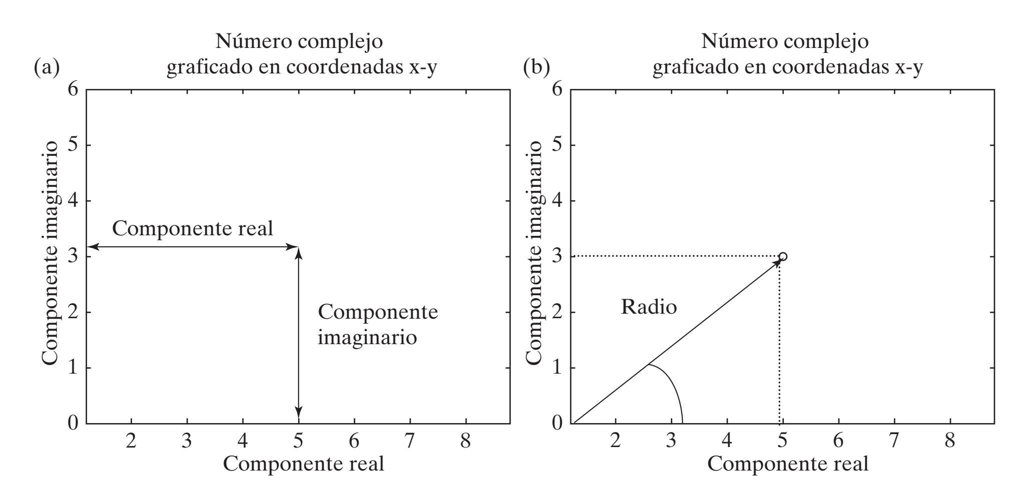


Figura 3.15
(a) Número complejo representado en un sistema de coordenadas cartesianas. (b) Un número complejo también se puede describir con coordenadas polares.

De igual modo, el ángulo se encuentra con la función **angle**:

```
angle(A)
ans =
    0.5404
```

El resultado se expresa en radianes. Ambas funciones, **abs** y **angle**, aceptarán como entrada escalares o arreglos. Recuerde que B es un arreglo 1 × 3 de números complejos:

```
B =
   5.0000 + 3.0000i   6.0000 + 3.0000i  15.0000 + 9.0000i
```

La función **abs** regresa el radio si el número se representa en coordenadas polares:

```
abs(B)
ans =
    5.8310    6.7082   17.4929
```

El ángulo desde la horizontal se puede encontrar con la función **angle**:

```
angle(B)
ans =
    0.5404    0.4636    0.5404
```

En la tabla 3.13 se resumen las funciones MATLAB usadas comúnmente con números complejos.

Tabla 3.13 Funciones usadas con números complejos

`abs(x)`	Calcula el valor absoluto de un número complejo mediante el teorema de Pitágoras. Esto es equivalente al radio si el número complejo se representa en coordenadas polares. Por ejemplo, si $x = 3 + 4i$, el valor absoluto es $\sqrt{3^2 + 4^2} = 5$	`x=3+4i;`
		`abs(x)`
		`ans =`
		`5`

`angle(x)`	Calcula el ángulo desde la horizontal, en radianes, cuando un número complejo se representa en coordenadas polares	`x=3+4i;`
		`angle(x)`
		`ans =`
		`0.9273`

`complex(x,y)`	Genera un número complejo con un componente real x y un componente imaginario y	`x=3;`
		`y=4;`
		`complex(x,y)`
		`ans =`
		`3.0000 +`
		`4.0000i`

`real(x)`	Extrae el componente real de un número complejo	`x=3+4i;`
		`real(x)`
		`ans =`
		`3`

`imag(x)`	Extrae el componente imaginario de un número complejo	`x=3+4i;`
		`imag(x)`
		`ans =`
		`4`

`isreal(x)`	Determina si los valores en un arreglo son reales. Si lo son, la función regresa 1; si son complejos, regresa 0	`x=3+4i;`
		`isreal(x)`
		`ans =`
		`0`

`conj(x)`	Genera la conjugada compleja de un número complejo	`x=3+4i;`
		`conj(x)`
		`ans =`
		`3.0000 - 4.0000i`

Ejercicio de práctica 3.10

1. Cree los siguientes números complejos:
   a. $A = 1 + i$
   b. $B = 2 - 3i$
   c. $C = 8 + 2i$
2. Cree un vector **D** de números complejos cuyos componentes reales son 2, 4 y 6 y cuyos componentes imaginarios son -3, 8 y -16.
3. Encuentre la magnitud (valor absoluto) de cada uno de los vectores que creó en los problemas 1 y 2.
4. Encuentre el ángulo desde la horizontal de cada uno de los números complejos que creó en los problemas 1 y 2.
5. Encuentre la conjugada compleja del vector **D**.
6. Use el operador transpuesto para encontrar la conjugada compleja del vector **D**.
7. Multiplique **A** por su conjugada compleja y luego saque la raíz cuadrada de su respuesta. ¿Cómo se compara este valor contra la magnitud (valor absoluto) de **A**?

3.8 LIMITACIONES COMPUTACIONALES

Las variables que se almacenan en una computadora pueden asumir un amplio rango de valores. En la mayoría de las computadoras, el rango se extiende desde aproximadamente 10^{-308} hasta 10^{308}, que debe ser suficiente para acomodar la mayoría de los cálculos. MATLAB incluye funciones para identificar los números reales más grandes y los enteros más grandes que el programa puede procesar (tabla 3.14).

Idea clave: existe un límite acerca de cuán pequeño o cuán grande puede ser un número manejado por los programas de cómputo.

Tabla 3.14 Límites computacionales

`realmax`	Regresa el número punto flotante más grande posible usado en MATLAB	`realmax`
		`ans =`
		`1.7977e+308`

`realmin`	Regresa el número punto flotante más pequeño posible usado en MATLAB	`realmin`
		`ans =`
		`2.2251e-308`

`intmax`	Regresa el número entero más grande posible usado en MATLAB	`intmax`
		`ans =`
		`2147483647`

`intmin`	Regresa el número entero más pequeño posible usado en MATLAB	`intmin`
		`ans =`
		`-2147483648`

El valor de **realmax** corresponde aproximadamente a 2^{1024}, un valor que resulta del hecho de que las computadoras en realidad realizan sus cálculos en aritmética binaria (base 2). Desde luego, es posible formular un problema en el que el resultado de una expresión sea más grande o más pequeño que el máximo permitido. Por ejemplo, suponga que se ejecutan los siguientes comandos:

```
x = 2.5e200;
y = 1.0e200;
z = x*y
```

desbordamiento: resultado de un cálculo que es demasiado grande para que el programa de cómputo lo pueda manejar

subdesbordamiento: resultado de un cálculo que es demasiado pequeño como para que la computadora lo distinga de cero

MATLAB responde con

```
z =
      Inf
```

puesto que la respuesta (**2.5*e400**) está fuera del rango permisible. Este error se llama *desbordamiento de exponente*, pues el exponente del resultado de una operación aritmética es demasiado grande para almacenarse en la memoria de la computadora.

El subdesbordamiento de exponente es un error similar, causado porque el exponente del resultado de una operación aritmética es demasiado *pequeño* como para almacenarse en la memoria de la computadora. Al usar el mismo rango permisible, se obtiene un subdesbordamiento de exponente con los siguientes comandos:

```
x = 2.5e-200;
y = 1.0e200
z = x/y
```

En conjunto, estos comandos regresan

```
z = 0
```

El resultado de un desbordamiento de exponente es cero.

Idea clave: la planeación cuidadosa le puede ayudar a evitar el desbordamiento o el subdesbordamiento en los cálculos.

También se sabe que la división entre cero es una operación inválida. Si una expresión resulta en una división entre cero, el resultado de la división es infinito:

```
z = y/0
z =
      Inf
```

MATLAB puede imprimir una advertencia que le diga que la división entre cero no es posible.

Al realizar cálculos con números o muy grandes o muy pequeños, puede ser posible reordenar los cálculos para evitar un subdesbordamiento o un desbordamiento. Suponga, por ejemplo, que le gustaría realizar la siguiente cadena de multiplicaciones:

$$(2.5 \times 10^{200}) \times (2 \times 10^{200}) \times (1 \times 10^{-100})$$

La respuesta es 5×10^{300}, dentro de los límites permitidos por MATLAB. Sin embargo, considere lo que ocurre cuando ingresa el problema en MATLAB:

```
2.5e200*2e200*1e-100
ans =
      Inf
```

Puesto que MATLAB ejecuta el problema de izquierda a derecha, la primera multiplicación produce un valor fuera del rango permisible (5×10^{400}), lo que resulta en una respuesta de infinito. Sin embargo, al reordenar el problema a

```
2.5e200*1e-100*2e200
ans =
5.0000e+300
```

se evita el desbordamiento y se encuentra la respuesta correcta.

3.9 VALORES ESPECIALES Y FUNCIONES VARIAS

La mayoría de las funciones, aunque no todas, requieren un argumento de entrada. Si bien se usan como si fuesen constantes escalares, las funciones que se mencionan en la tabla 3.15 *no* requieren entrada alguna.

Tabla 3.15 Funciones especiales

`pi`	Constante matemática π	`pi`
		`ans =`
		`3.1416`

`i`	Número imaginario	`i`
		`ans =`
		`0 + 1.0000i`

`j`	Número imaginario	`j`
		`ans =`
		`0 + 1.0000i`

`Inf`	Infinito, que con frecuencia ocurre durante un desbordamiento de cálculo o cuando un número se divide entre cero	`5/0`
		`Warning: Divide by zero.`
		`ans =`
		`Inf`

`NaN`	No es un número
	Ocurre cuando un cálculo es indefinido
		`0/0`
		`Warning: Divide by zero.`
		`ans =`
		`NaN`
		
		`inf/inf`
		`ans =`
		`NaN`

`clock`	Tiempo actual.
	Regresa un arreglo de seis miembros [año, mes, día, hora, minuto, segundo]. Cuando la función clock se solicitó el 6 de enero de 2006, a las 12:07 A.M. y 8.7 segundos, MATLAB regresó la salida que se muestra a la derecha. Las funciones **fix** y **clock** juntas resultan en un formato que es más fácil de leer. La función **fix** redondea hacia cero. Un resultado similar se podría obtener al establecer **format bank**
		`clock`
		`ans =`
		`1.0e+003 *`
		`2.0060 0.0010 0.0060`
		`0.0120 0.0070 0.0087`
		
		`fix(clock)`
		
		`ans =`
		`2006 1 6`
		`12 7 8`

`date`	Fecha actual.
	Similar a la función clock. Sin embargo, regresa la fecha en un "formato de cadena"
		`date`
		`ans =`
		`06-Jan-2006`

`eps`	La distancia entre 1 y el siguiente número punto flotante de doble precisión más grande	`eps`
		`ans =`
		`2.2204e-016`

MATLAB le permite redefinir estos valores especiales como nombres de variable; sin embargo, hacerlo puede tener consecuencias inesperadas. Por ejemplo, se permite el siguiente código MATLAB, aunque no es aconsejable:

```
pi = 12.8;
```

A partir de este punto, cada vez que se pida la variable **pi**, se usará el nuevo valor. De igual modo, puede redefinir cualquier función como un nombre de variable, tal como

```
sin = 10;
```

Para restaurar **sin** a su empleo como función trigonométrica (o para restaurar el valor por defecto de **pi**), debe limpiar el área de trabajo con

```
clear
```

Compruebe ahora el resultado al escribir el comando para π.

```
pi
```

Este comando regresa

```
pi =
      3.1416
```

Sugerencia

La función *i* es la más común de estas funciones que, de manera no intencional, renombran los usuarios de MATLAB.

Ejercicio de práctica 3.11

1. Use la función clock para agregar la hora y fecha a su hoja de trabajo.
2. Use la función date para agregar la fecha a su hoja de trabajo.
3. Convierta los siguientes cálculos a código MATLAB y explique sus resultados:

   a. 322! (Recuerde que ! significa factorial para un matemático.)
   b. 5×10^{500}
   c. $1/5 \times 10^{500}$
   d. 0/0

RESUMEN

En este capítulo se exploraron varias funciones predefinidas de MATLAB, incluidas las siguientes:

- funciones matemáticas generales, como
  - funciones exponenciales.
  - funciones logarítmicas.
  - raíces.
- funciones de redondeo
- funciones usadas en matemáticas discretas, tales como
  - funciones de factorización.
  - funciones de números primos.

- funciones trigonométricas, incluidas
  - funciones trigonométricas estándar.
  - funciones trigonométricas inversas.
  - funciones trigonométricas hiperbólicas.
  - funciones trigonométricas que usan grados en lugar de radianes.
- funciones de análisis de datos, tales como
  - máximos y mínimos.
  - promedios (media y mediana).
  - sumas y productos.
  - ordenamiento.
  - desviación estándar y varianza.
- generación de números aleatorios para
  - distribuciones uniformes.
  - distribuciones gaussianas (normales).
- funciones usadas con números complejos

Se exploraron los límites computacionales inherentes a MATLAB y se introdujeron valores especiales, como **pi**, que son internos al programa.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona y describe brevemente todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres y funciones especiales	
`eps`	reconoce diferencia más pequeña
`i`	número imaginario
`clock`	regresa la hora
`date`	regresa la fecha
`Inf`	infinito
`intmax`	regresa el número entero más grande posible usado en MATLAB
`intmin`	regresa el número entero más pequeño posible usado en MATLAB
`j`	número imaginario
`NaN`	no es un número
`pi`	constante matemática π
`realmax`	regresa el número punto flotante más grande posible usado en MATLAB
`realmin`	regresa el número punto flotante más pequeño posible usado en MATLAB

Comandos y funciones	
`abs`	calcula el valor absoluto de un número real o la magnitud de un número complejo
`angle`	calcula el ángulo cuando los números complejos se representan en coordenadas polares
`asin`	calcula el seno inverso (arcoseno)
`asind`	calcula el seno inverso y reporta el resultado en grados
`ceil`	redondea al entero más cercano hacia infinito positivo
`complex`	crea un número complejo
`conj`	crea la conjugada compleja de un número complejo
`cos`	calcula el coseno
`cumprod`	calcula un producto acumulado de los valores en un arreglo
`cumsum`	calcula una suma acumulada de los valores en un arreglo
`erf`	calcula la función error
`exp`	calcula el valor de e^x

(*Continúa*)

Comandos y funciones (continuación)	
`factor`	encuentra los factores primos
`factorial`	calcula el factorial
`fix`	redondea al entero más cercano hacia cero
`floor`	redondea hacia el entero más cercano hacia menos infinito
`gcd`	encuentra el máximo común denominador
`help`	abre la función help
`helpwin`	abre la función help en ventana
`imag`	extrae el componente imaginario de un número complejo
`isprime`	determina si un valor es primo
`isreal`	determina si un valor es real o complejo
`lcn`	encuentra el mínimo común denominador
`length`	determina la mayor dimensión de un arreglo
`log`	calcula el logaritmo natural o el logaritmo a la base *e* ($\log_e$)
`log10`	calcula el logaritmo común o el logaritmo a la base 10 ($\log_{10}$)
`log2`	calcula el logaritmo a la base 2 ($\log_2$)
`max`	encuentra el valor máximo en un arreglo y determina cuál elemento almacena el valor máximo
`mean`	calcula el promedio de los elementos en un arreglo
`median`	encuentra la mediana de los elementos en un arreglo
`min`	encuentra el valor mínimo en un arreglo y determina cuál elemento almacena el valor mínimo
`nthroot`	encuentra la *n*-ésima raíz real de la matriz de entrada
`primes`	encuentra los números primos menores que el valor de entrada
`prod`	multiplica los valores en un arreglo
`rand`	calcula números aleatorios distribuidos de manera pareja
`randn`	calcula números aleatorios distribuidos de manera normal (gaussiana)
`rats`	convierte la entrada a una representación racional (es decir, una fracción)
`real`	extrae el componente real de un número complejo
`rem`	calcula el residuo en un problema de división
`round`	redondea al entero más cercano
`sign`	determina el signo (positivo o negativo)
`sin`	calcula el seno con radianes como entrada
`sind`	calcula el seno con ángulos en grados como entrada
`sinh`	calcula el seno hiperbólico
`size`	determina el número de filas y columnas en un arreglo
`sort`	ordena los elementos de un vector
`sortrows`	ordena las filas de un vector sobre la base de los valores en la primera columna
`sound`	toca archivos de música
`sqrt`	calcula la raíz cuadrada de un número
`std`	determina la desviación estándar
`sum`	suma los valores en un arreglo
`tan`	calcula la tangente con radianes como entrada
`var`	calcula la varianza

TÉRMINOS CLAVE

anidado
argumento
desbordamiento
desviación estándar
entrada de función
función
matemáticas discretas
media
mediana
número aleatorio uniforme
números complejos
números racionales
números reales
promedio
semilla
subdesbordamiento
variación aleatoria gaussiana
variación aleatorio normal
varianza

PROBLEMAS

Funciones matemáticas elementales

3.1 Encuentre la raíz cúbica de -5, tanto con la función **nthroot** como con elevar -5 a la potencia 1/3. Explique la diferencia en sus respuestas. Pruebe que ambos resultados de hecho son respuestas correctas al elevarlos al cubo y mostrar que son iguales a -5.

3.2 MATLAB contiene funciones para calcular el logaritmo natural (**log**), el logaritmo a la base 10 (**log10**) y el logaritmo a la base 2 (**log2**). Sin embargo, si quiere encontrar un logaritmo de base distinta (por ejemplo, base *b*), tendrá que hacer la matemática por usted mismo con la fórmula

$$\log_b(x) = \frac{\log_e(x)}{\log_e(b)}$$

¿Cuál es el $\log_b$ de 10 cuando *b* se define de 1 a 10 en incrementos de 1?

3.3 Las poblaciones tienden a expandirse exponencialmente. Esto es

$$P = P_0 e^{rt}$$

donde

P = población actual,
P_0 = población original,
r = tarifa de crecimiento continua, expresado como fracción, y
t = tiempo.

Si originalmente se tienen 100 conejos que se reproducen a una tasa de crecimiento constante de 90% ($r = 0.9$) por año, encuentre cuántos conejos tendrá al final de 10 años.

3.4 Las tasas de reacción química son proporcionales a una constante de tasa *k* que cambia con la temperatura de acuerdo con la ecuación Arrhenius

$$k = k_0 e^{-Q/RT}$$

Para cierta reacción

$$Q = 8000 \text{ cal/mol}$$
$$R = 1.987 \text{ cal/mol K}$$
$$k_0 = 1200 \text{ min}^{-1}$$

Encuentre los valores de *k* para temperaturas desde 100 K hasta 500 K, en incrementos de 50 grados. Cree una tabla con sus resultados.

3.5 Considere los requerimientos de aire acondicionado de la gran casa que se muestra en la figura P3.5. El interior de la casa se calienta por calor que se desecha de la iluminación y los aparatos eléctricos, del calor que se filtra desde el exterior y del calor que expulsan las personas en la casa. Un acondicionador de aire debe ser capaz de remover toda esta energía térmica con la finalidad de evitar que aumente la temperatura interna. Suponga que hay 20 focos que expulsan 100 J/s de energía cada uno y cuatro aparatos que expulsan 500 J/s cada uno. Suponga también que el calor se filtra del exterior a una tasa de 3000 J/s.

(a) ¿Cuánto calor por segundo debe remover de la casa el acondicionador de aire?

(b) Una unidad particular de acondicionador de aire puede manipular 2000 J/s. ¿Cuántas de estas unidades se requieren para mantener constante la temperatura de la casa?

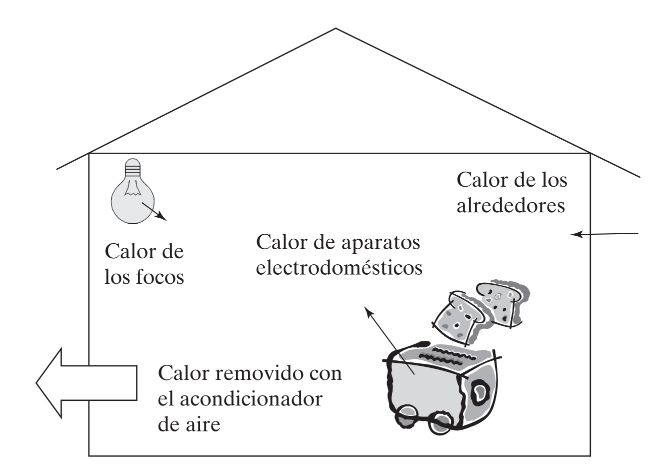


Figura P3.5
El acondicionador de aire debe remover calor de varias fuentes.

3.6 Muchos problemas que involucran probabilidad se pueden resolver con factoriales. Por ejemplo, el número de formas en que se pueden ordenar cinco cartas es $5 \times 4 \times 3 \times 2 \times 1 = 5! = 120$. Cuando selecciona la primera carta, tiene cinco opciones; cuando selecciona la segunda carta, tiene sólo cuatro opciones restantes, luego tres, dos y una. Este enfoque se llama matemática combinatoria.

(a) Si tiene cuatro personas, ¿en cuántas formas diferentes puede ordenarlas en una línea?
(b) Si tiene 10 baldosas diferentes, ¿en cuántas formas diferentes puede ordenarlas?

3.7 Si tiene cuatro personas, ¿cuántos diferentes comités de dos personas puede crear? Recuerde que un comité de Bob y Alice es el mismo que un comité de Alice y Bob.

3.8 Existen 52 cartas *diferentes* en un mazo. ¿Cuántas posibles manos diferentes de 5 cartas existen? Recuerde: cada mano se puede ordenar de 120 formas diferentes.

3.9 Los números primos muy grandes se usan en criptografía. ¿Cuántos números primos existen entre 10,000 y 20,000? (Éstos no son números primos suficientemente grandes como para usarse en cifrados.) (*Sugerencia:* use la función **primes** y el comando **length**.)

Funciones trigonométricas

3.10 A veces es conveniente tener una tabla de seno, coseno y tangente en lugar de usar una calculadora. Cree una tabla de estas tres funciones trigonométricas para ángulos de 0 a 2π, con un espaciamiento de 0.1 radianes. Su tabla debe contener una columna para el ángulo y luego el seno, coseno y tangente.

3.11 El desplazamiento del resorte oscilatorio que se muestra en la figura P3.11 se puede describir mediante

$$x = A \cos(\omega t)$$

donde

x = desplazamiento en el tiempo t,
A = desplazamiento máximo,
ω = frecuencia angular, que depende de la constante de resorte y la masa unida al mismo, y
t = tiempo.

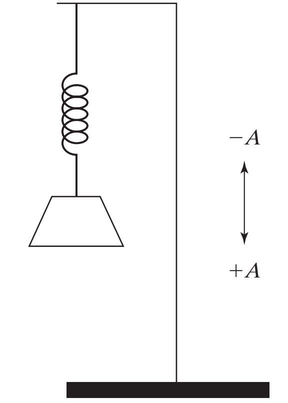


Figura P3.11
Resorte oscilatorio.

Encuentre el desplazamiento x para tiempos desde 0 hasta 10 segundos cuando el desplazamiento máximo A es 4 cm y la frecuencia angular es 0.6 radianes/s. Presente sus resultados en una tabla de desplazamiento y tiempo.

3.12 La aceleración del resorte descrito en el problema anterior es

$$a = -A\omega^2 \cos(\omega t)$$

Encuentre la aceleración para tiempos desde 0 hasta 10 segundos, con los valores constantes del problema anterior. Cree una tabla que incluya el tiempo, el desplazamiento de valores correspondientes en el problema anterior y la aceleración.

3.13 Puede usar trigonometría para encontrar la altura de un edificio, como se muestra en la figura P3.13. Suponga que mide el ángulo entre la línea de visión y la línea horizontal que conecta el punto de medición y el edificio. Puede calcular la altura del edificio con las siguientes fórmulas:

$$\tan(\theta) = h/d$$
$$h = d\tan(\theta)$$

Suponga que la distancia al edificio, a lo largo del suelo, es de 120 m y que el ángulo medido a lo largo de la línea de visión es $30° \pm 3°$. Encuentre las alturas máxima y mínima que puede tener el edificio.

3.14 Considere el edificio del problema anterior.

(a) Si tiene 20 pies de alto y usted está a 20 pies de distancia, ¿a qué ángulo del suelo tendrá que inclinar su cabeza para ver la punta del edificio? (Suponga que su cabeza está a la par con el suelo.)

(b) ¿Qué distancia hay desde su cabeza hasta la punta del edificio?

Funciones de análisis de datos

3.15 Considere la siguiente tabla de datos que representan lecturas de temperatura en un reactor:

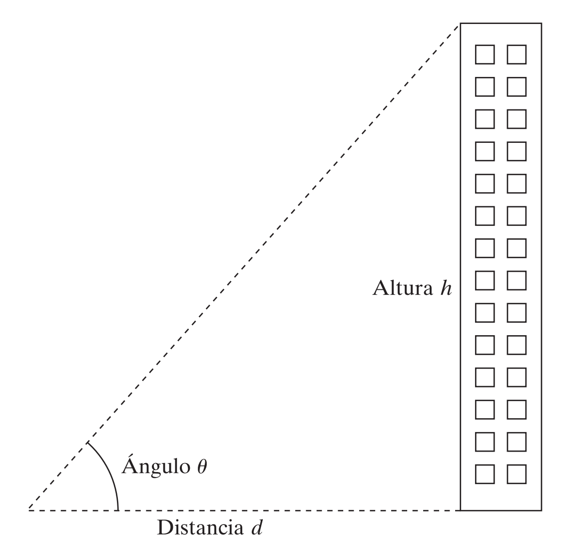


Figura P3.13
Puede determinar la altura de un edificio con trigonometría.

Termocople 1	Termocople 2	Termocople 3
84.3	90.0	86.7
86.4	89.5	87.6
85.2	88.6	88.3
87.1	88.9	85.3
83.5	88.9	80.3
84.8	90.4	82.4
85.0	89.3	83.4
85.3	89.5	85.4
85.3	88.9	86.3
85.2	89.1	85.3
82.3	89.5	89.0
84.7	89.4	87.3
83.6	89.8	87.2

Su instructor le puede proporcionar un archivo llamado **thermocouple.dat**, o es posible que usted tenga que ingresar los datos.

Use MATLAB para encontrar:

(a) la temperatura máxima medida por cada termocople.
(b) la temperatura mínima medida por cada termocople.

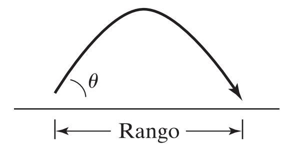


Figura P3.16
El rango depende del ángulo de lanzamiento y la velocidad de lanzamiento.

3.16 El rango de un objeto que se dispara en un ángulo θ con respecto al eje x y una velocidad inicial v_0 (figura P3.16) está dado por

$$\text{Rango} = \frac{v_0^2}{g}\,\text{sen}(2\theta)$$

para $0 \le \theta \le \pi/2$ y resistencia del aire despreciable. Use $g = 9.81\ \text{m/s}^2$ y una velocidad inicial v_0 de 100 m/s. Muestre que el rango máximo se obtiene a aproximadamente $\theta = \pi/4$ al calcular el rango en incrementos de 0.05 entre $0 \le \theta \le \pi/2$. No podrá encontrar el ángulo exacto que produce el rango máximo, porque sus cálculos están en ángulos igualmente espaciados de 0.05 radianes.

3.17 El vector

$$G = [68, 83, 61, 70, 75, 82, 57, 5, 76, 85, 62, 71, 96, 78, 76, 68, 72, 75, 83, 93]$$

representa la distribución de calificaciones finales en un curso de dinámica. Calcule la media, mediana y la desviación estándar de G. ¿Cuál representa mejor la “calificación más usual”, la media o la mediana? ¿Por qué? Use MATLAB para determinar el número de calificaciones en el arreglo (no sólo las cuente) y ordénelas en orden ascendente.

3.18 Genere 10,000 números aleatorios gaussianos con una media de 80 y desviación estándar de 23.5. (Querrá suprimir la salida de modo que no abrume la ventana de comandos con datos.) Use la función **mean** para confirmar que su arreglo en realidad tiene una media de 80. Use la función **std** para confirmar que su desviación estándar realmente es 23.5.

3.19 Use la función **date** para agregar la fecha actual a su tarea.

Números aleatorios

3.20 Muchos juegos requieren que el jugador tire dos dados. El número en cada dado puede variar entre 1 y 6.

(a) Use la función **rand** en combinación con una función de redondeo para crear una simulación de una tirada de un dado.
(b) Use sus resultados de la parte (a) para crear una simulación del valor de tirar un segundo dado.
(c) Sume sus dos resultados para crear un valor que represente la tirada total durante cada turno.
(d) Use su programa para determinar los valores de tirada en un juego de mesa de su preferencia, o use el juego que se muestra en la figura P3.20.

3.21 Suponga que diseña un contenedor para embarcar materiales médicos sensibles entre hospitales. El contenedor necesita mantener los contenidos dentro de un rango de temperatura específico. Usted crea un modelo que predice cómo responde el contenedor a la temperatura exterior y ahora necesita correr una simulación.

(a) Cree una distribución normal (distribución gaussiana) de temperaturas con una media de 70 °F y una desviación estándar de 2°, que corresponde a una duración de 2 horas. Necesitará una temperatura para cada valor de tiempo desde 0 hasta 120 minutos. (Éstos son 121 valores.)
(b) Grafique los datos en una gráfica *x-y*. No se preocupe por las etiquetas. Recuerde que la función MATLAB para graficación es **plot(x,y)**.
(c) Encuentre la temperatura máxima, la temperatura mínima y los tiempos en que ellas ocurren.

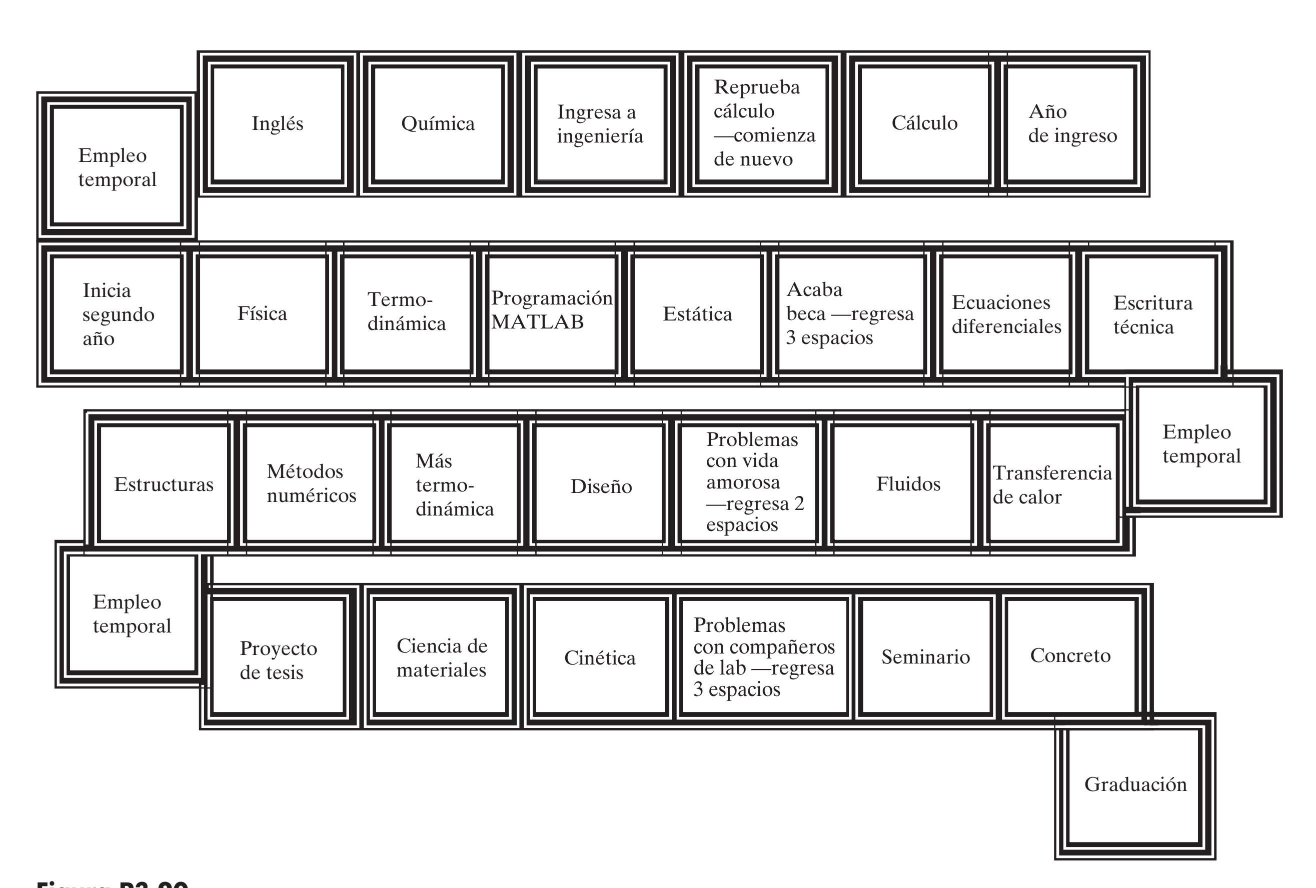


Figura P3.20
El juego de la universidad.

CAPÍTULO

4

Manipulación de matrices MATLAB

Objetivos

Después de leer este capítulo, el alumno será capaz de

- manipular matrices.
- extraer datos de matrices.
- resolver problemas con dos variables matriz de diferente tamaño.
- crear y usar matrices especiales.

4.1 MANIPULACIÓN DE MATRICES

Conforme resuelva problemas cada vez más complicados con MATLAB, encontrará que necesitará combinar pequeñas matrices con matrices más grandes, extraer información de matrices grandes, crear matrices muy grandes y usar matrices con propiedades especiales.

4.1.1 Definición de matrices

En MATLAB, una matriz se puede definir al escribir una lista de números encerrada entre corchetes. Los números se pueden separar mediante espacios o comas, a discreción del usuario. (Incluso puede combinar las dos técnicas en la misma definición de matriz.) Las nuevas filas se indican con punto y coma. Por ejemplo,

```
A = [3.5];
B = [1.5, 3.1]; o  B = [1.5   3.1];
C = [-1, 0, 0; 1, 1, 0; 0, 0, 2];
```

También se puede definir una matriz al hacer una lista de cada fila en una línea separada, como en el siguiente conjunto de comandos MATLAB:

```
C =    [-1,  0, 0;
         1,  1, 0;
         1, -1, 0;
         0,   0, 2]
```

Incluso no necesita ingresar el punto y coma para indicar una nueva fila. MATLAB interpreta

```
C =    [-1,  0, 0
         1,  1, 0
         1, -1, 0
         0,   0, 2]
```

como una matriz 4 × 3. También podría ingresar una matriz columna de esta forma:

```
A =
    1
    2
    3
```

elipsis: conjunto de tres periodos que se usa para indicar que una fila continúa en la siguiente línea

Si existen demasiados números en una fila como para encajar en una línea, puede continuar el enunciado en la línea siguiente, pero se requieren una coma y una elipsis (...) al final de la línea, lo que indica que la fila continúa. También puede usar la elipsis para continuar otros enunciados de asignación largos en MATLAB.

Si quiere definir **F** con 10 valores, se podría usar cualquiera de los siguientes enunciados:

```
F = [1, 52, 64, 197, 42, -42, 55, 82, 22, 109];  o
F = [1, 52, 64, 197, 42, -42, ...
        55, 82, 22, 109];
```

MATLAB también le permite definir una matriz en términos de otra matriz que ya se haya definido. Por ejemplo, los enunciados

```
B = [1.5, 3.1];
S = [3.0, B]
```

regresa

```
S =
    3.0    1.5   3.1
```

De manera similar,

```
T = [ 1, 2, 3; S]
```

regresa

```
T =
      1      2      3
      3      1.5    3.1
```

índice: número que se usa para identificar elementos en un arreglo

Se pueden cambiar los valores en una matriz, o incluir valores adicionales, con un número índice para especificar un elemento particular. Este proceso se llama *indexación en un arreglo*. Por tanto, el comando

```
S(2) = -1.0;
```

cambia el segundo valor en la matriz **S** de 1.5 a −1. Si escribe el nombre de matriz

```
S
```

en la ventana de comandos, entonces MATLAB regresa

```
S =
      3.0    -1.0   3.1
```

También se puede extender una matriz al definir nuevos elementos. Si ejecuta el comando

```
S(4) = 5.5;
```

se extiende la matriz **S** a cuatro elementos en lugar de tres. Si se define el elemento

```
S(8) = 9.5;
```

la matriz **S** tendrá ocho valores, y los valores de **S(5)**, **S(6)** y **S(7)** se establecerá a 0. En consecuencia,

```
S
```

regresa

```
S =
      3.0    -1.0   3.1    5.5    0      0      0      9.5
```

4.1.2 Uso del operador dos puntos

El operador dos puntos es un operador muy poderoso para definir nuevas matrices y modificar las existentes. Primero, puede definir una matriz igualmente espaciada con el operador dos puntos. Por ejemplo,

```
H = 1:8
```

regresa

```
H =
      1      2      3      4      5      6      7      8
```

El espaciamiento por defecto es 1. Sin embargo, cuando se usan los dos puntos para separar tres números, el valor medio se convierte en el espaciamiento. Por tanto,

```
time = 0.0 : 0.5 : 2.0
```

regresa

```
time =
        0    0.5000    1.0000    1.5000    2.0000
```

El operador dos puntos también se puede usar para extraer datos de las matrices, una característica que es muy útil en análisis de datos. Cuando en una matriz se usan dos puntos como referencia en lugar de un número índice específico, los dos puntos representan toda la fila o columna.

```
M =   [1 2 3 4 5;
       2 3 4 5 6;
       3 4 5 6 7];
```

Se puede extraer la columna 1 de la matriz **M** con el comando

```
x = M(:, 1)
```

lo que regresa

```
x =
      1
      2
      3
```

Esta sintaxis se puede leer como "todas las filas en la columna 1". Puede extraer cualquiera de las columnas en una forma similar. Por ejemplo,

```
y = M(:, 4)
```

regresa

```
y =
        4
        5
        6
```

y se puede interpretar como "todas las filas en la columna 4". De igual modo, para extraer una fila,

```
z = M(1,:)
```

regresa

```
z =
        1        2        3        4        5
```

y se lee como "fila 1, todas las columnas".

No tiene que extraer toda una fila o toda una columna. El operador dos puntos también se puede usar para significar "desde fila _ hasta fila _" o "desde columna _ hasta columna _". Para extraer las dos filas inferiores de la matriz **M**, escriba

```
w = M(2:3,:)
```

que regresa

```
w =
      2      3      4      5      6
      3      4      5      6      7
```

y se lee "filas 2 a 3, todas las columnas". De manera similar, para extraer sólo los cuatro números en la esquina inferior derecha de la matriz **M**,

```
w=M(2:3,4:5)
```

regresa

```
w =
      5      6
      6      7
```

y lee "filas 2 a 3 en las columnas 4 a 5".

En MATLAB, es válido tener una matriz que esté vacía. Por ejemplo, los siguientes enunciados generarán cada uno una matriz vacía:

```
a = [ ];
b = 4:-1:5;
```

Finalmente, usar el nombre de matriz con un solo dos puntos, como

```
M(:)
```

transforma la matriz en una larga columna.

```
M =
  1
  2
  3
  2
  3
  4
  3
  4
  5
  4
  5
  6
  5
  6
  7
```

La matriz se formó al listar primero la columna 1, luego agregar la columna 2 al final, tomar la columna 3, etcétera. En realidad, la computadora no almacena arreglos bidimensionales en un patrón bidimensional. Más bien, "piensa" en una matriz como en una larga lista, tal como la matriz **M** a la izquierda: al usar un solo número índice o al usar la notación fila, columna. Para encontrar el valor en la fila 2, columna 3, use los siguientes comandos:

Idea clave: puede identificar un elemento con el uso de un solo número o índices que representen la fila y columna.

```
M
M =
     1     2     3     4     5
     2     3     4     5     6
     3     4     5     6     7
M(2,3)
ans =
           4
```

De manera alternativa, puede usar un solo número índice. El valor en la fila 2, columna 3 de la matriz **M** es el elemento número 8. (Cuente la columna 1, luego la columna 2 y finalmente en la columna 3 hasta el elemento correcto.) El comando MATLAB asociado es

```
M(8)
ans = 4
```

Sugerencia

Puede usar la palabra "end" para identificar la fila o columna final en una matriz, incluso si no sabe qué tan grade es. Por ejemplo,

```
M(1,end)
```

regresa

```
M(1,end)
ans =
     5
```

y

```
M(end, end)
```

regresa

```
ans =
     7
```

como lo hace

```
M(end)
ans =
     7
```

Ejercicio de práctica 4.1

Cree variables MATLAB para representar las siguientes matrices y úselas en los ejercicios que siguen:

$$a = [12 \quad 17 \quad 3 \quad 6] \qquad b = \begin{bmatrix} 5 & 8 & 3 \\ 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix} \qquad c = \begin{bmatrix} 22 \\ 17 \\ 4 \end{bmatrix}$$

1. Asigne a la variable **x1** el valor en la segunda columna de la matriz *a*. En ocasiones, esto se representa en los libros de matemáticas como el elemento $a_{1,2}$ y se podría expresar como **x1** = $a_{1,2}$.
2. Asigne a la variable **x2** la tercera columna de la matriz *b*.
3. Asigne a la variable **x3** la tercera fila de la matriz *b*.
4. Asigne a la variable **x4** los valores en la matriz *b* a lo largo de la diagonal (es decir: elementos $b_{1,1}$, $b_{2,2}$ y $b_{2,3}$).
5. Asigne a la variable **x5** los primeros tres valores en la matriz *a* como la primera fila y todos los valores en la matriz *b* como la segunda a la cuarta filas.
6. Asigne a la variable **x6** los valores en la matriz *c* como la primera columna, los valores en la matriz *b* como las columnas 2, 3 y 4, y los valores en la matriz *a* como la última fila.
7. Asigne a la variable **x7** el valor del elemento 8 en la matriz *b*, use el esquema de identificación de número de índice sencillo.
8. Convierta la matriz *b* en un vector columna llamado **x8**.

EJEMPLO 4.1

Uso de datos de temperatura

Los datos recopilados por el National Weather Service son extensivos, pero no siempre organizados exactamente en la forma que se quisiera (figura 4.1). Tome, por ejemplo, el resumen 1999 de los datos climatológicos de Asheville, Carolina del Norte. Estos datos se usarán para practicar la manipulación de matrices: extraer y recombinar elementos para formar nuevas matrices.

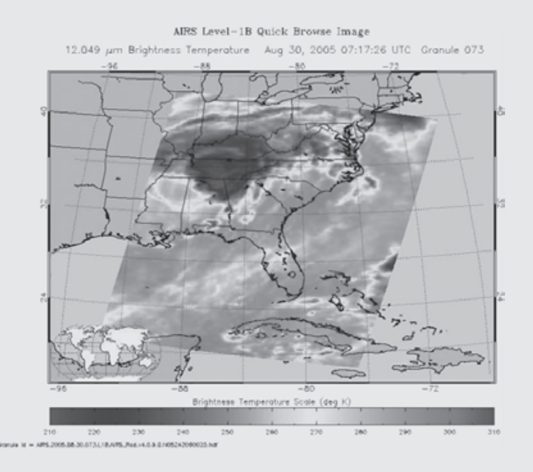


Figura 4.1
Datos de temperatura recopilados de un satélite climático usado para crear esta imagen en falso color. (Cortesía de NASA/Jet Propulsion Laboratory.)

La información numérica se extrajo de la tabla y está en un archivo Excel llamado **Asheville_1999.xls** (Apéndice C). Use MATLAB para conformar que los valores reportados en la fila anual son correctos para la temperatura máxima media y la temperatura mínima media, así como para la temperatura anual alta y la temperatura anual baja. Combine estas cuatro columnas de datos en una nueva matriz llamada **temp_data**.

1. Establezca el problema.
   Calcule la temperatura anual máxima media, la temperatura anual mínima media, la temperatura más alta alcanzada durante el año y la temperatura más baja alcanzada durante el año para 1999 en Asheville, Carolina del Norte.

2. Describa las entradas y salidas.

   Entrada Importe una matriz desde el archivo Excell **Asheville_1999.xls**

   Salida Encuentre los siguientes cuatro valores: temperatura anual máxima media
   temperatura anual mínima media
   temperatura más alta
   temperatura más baja

   Cree una matriz compuesta de los valores de temperatura máxima media, los valores de temperatura mínima media, las temperaturas mensuales más altas y las temperaturas mensuales más bajas. No incluya los datos anuales.

3. Desarrolle un ejemplo a mano.
   Use una calculadora, sume los valores en la columna 2 de la tabla y divida entre 12.

4. Desarrolle una solución MATLAB.
   Primero importe los datos de Excel y luego guárdelos en el directorio actual como **Asheville_1999**. Guarde la variable **Asheville_1999** como el archivo **Asheville_1999.mat**. Esto lo hace disponible para que se cargue en el área de trabajo desde el programa archivo-m:

```
%  Ejemplo 4.1
%  En este ejemplo se extraen datos de una matriz grande y
%  se usan funciones de análisis de datos para encontrar
   temperaturas
%  alta media y baja media para el año y para encontrar la
%  temperatura alta y la temperatura baja para el año
%
clear, clc
%  carga la matriz de datos desde un archivo
load asheville_1999
%  extrae las temperaturas alta media de la matriz grande
mean_max = asheville_1999(1:12,2);
%  extrae las temperaturas baja media de la matriz grande
mean_min = asheville_1999(1:12,3);
%  Calcula las medias anuales
annual_mean_max = mean(mean_max)
annual_mean_min = mean(mean_min)
%  extrae las temperaturas alta y baja de la matriz
%  grande
high_temp = asheville_1999(1:12,8);
low_temp = asheville_1999(1:12,10);
%  Encuentra las temperaturas máx y mín para el año
max_high = max(high_temp)
min_low = min(low_temp)
%  Crea una nueva matriz sólo con la información
%  de temperatura
new_table =[mean_max, mean_min, high_temp, low_temp]
```

Los resultados se despliegan en la ventana de comandos:

```
annual_mean_max =
   68.0500
annual_mean_min =
   46.3250
max_high =
    96
min_low =
     9
new_table =
   51.4000   31.5000   78.0000    9.0000
   52.6000   32.1000   66.0000   16.0000
   52.7000   32.5000   76.0000   22.0000
   70.1000   48.2000   83.0000   34.0000
   75.0000   51.5000   83.0000   40.0000
   80.2000   60.9000   90.0000   50.0000
   85.7000   64.9000   96.0000   56.0000
   86.4000   63.0000   94.0000   54.0000
   79.1000   54.6000   91.0000   39.0000
   67.6000   45.5000   78.0000   28.0000
   62.2000   40.7000   76.0000   26.0000
   53.6000   30.5000   69.0000   15.0000
```

5. Ponga a prueba la solución.
   Compare los resultados contra la línea inferior de la tabla de la Encuesta Climatológica de Asheville, Carolina del Norte. Es importante confirmar que los resultados son precisos antes de comenzar a usar cualquier programa de computadora para procesar datos.

4.2 PROBLEMAS CON DOS VARIABLES

Todos los cálculos realizados hasta el momento han usado sólo una variable. Desde luego, la mayoría de los fenómenos físicos puede variar con muchos factores diferentes. En esta sección se considera cómo realizar los mismos cálculos cuando las variables se representan mediante vectores.

Considere los siguientes enunciados MATLAB:

```
x = 3;
y = 5;
A = x * y
```

Dado que **x** y **y** son escalares, es un cálculo sencillo: **x** · **y** = 15, o

```
A =
       15
```

Ahora vea lo que ocurre si **x** es una matriz y **y** todavía es un escalar:

```
x = 1:5;
```

regresa cinco valores de **x**. Dado que **y** todavía es un escalar con sólo un valor (5),

```
A = x * y
```

regresa

```
A =
        5      10      15      20      25
```

Todo esto todavía es revisión. Pero, ¿qué ocurre si ahora **y** es un vector? Entonces

```
y = 1:3;
A = x * y
```

regresa un enunciado de error

```
??? Error using ==> *
Inner matrix dimensions must agree.
```

Este enunciado de error (las dimensiones internas de matriz deben concordar) le recuerda que el asterisco es el operador para multiplicación matricial, que no es lo que se quiere. Se quiere el operador punto-asterisco (.*), que realizará una multiplicación elemento por elemento. Sin embargo, los dos vectores, **x** y **y**, necesitarán tener la misma longitud para este propósito. En consecuencia,

Idea clave: cuando se formulan problemas con dos variables, las dimensiones de la matriz deben concordar.

```
y = linspace(1,3,5)
```

crea un nuevo vector **y** con cinco elementos igualmente espaciados:

```
y =
        1.0000    1.5000    2.0000    2.5000    3.0000
A = x .* y
A =
        1     3     6    10    15
```

No obstante, aunque esta solución funciona, el resultado probablemente no es lo que en realidad quiere. Puede pensar en los resultados como en la diagonal en una matriz (tabla 4.1).

¿Y si quiere conocer el resultado para el elemento 3 del vector **x** y el elemento 5 del vector **y**? Obviamente, este enfoque no da todas las posibles respuestas. Se quiere una matriz bidimensional de respuestas que corresponda a todas las combinaciones de **x** y **y**. Con la finalidad de que su respuesta, **A**, sea una matriz bidimensional, los vectores de entrada deben ser matrices bidimensionales. MATLAB tiene una función interna llamada **meshgrid**, que le ayudará a lograr esto, e incluso **x** y **y** no tienen que ser del mismo tamaño.

Primero cambie **y** de nuevo a un vector de tres elementos:

```
y = 1:3;
```

Tabla 4.1 Resultados de un cálculo elemento por elemento

		x				
		1	**2**	**3**	**4**	**5**
	1.0	1				
	1.5		3			
y	**2.0**			6		
	2.5				10	
	3.0			?		15

Luego se usará **meshgrid** para crear una nueva versión bidimensional de **x** y **y** que se llamarán **new_x** y **new_y**:

```
[new_x, new_y]=meshgrid(x,y)
```

Idea clave: use la función meshgrid para mapear dos variables unidimensionales en variables bidimensionales de igual tamaño.

El comando **meshgrid** toma los dos vectores de entrada y crea dos matrices bidimensionales. Cada una de las matrices resultantes tiene el mismo número de filas y columnas. El número de columnas se determina por el número de elementos en el vector **x**, y el número de filas se determina mediante el número de elementos en el vector **y**. Esta operación se llama *mapeo de vectores en un arreglo bidimensional*:

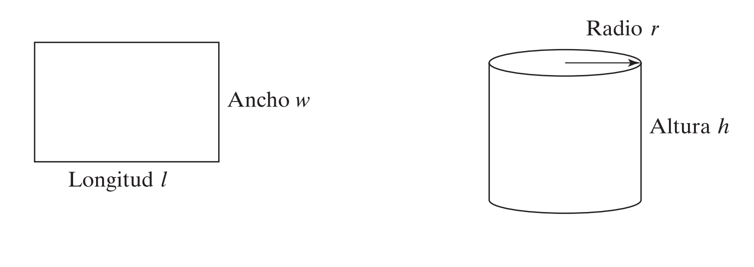
```
new_x =
        1     2     3     4     5
        1     2     3     4     5
        1     2     3     4     5
new_y =
        1     1     1     1     1
        2     2     2     2     2
        3     3     3     3     3
```

Note que todas las filas en **new_x** son las mismas y todas las columnas en **new_y** son las mismas. Ahora es posible multiplicar **new_x** por **new_y** y obtener la retícula bidimensional de resultados que realmente se quiere:

```
A = new_x.*new_y
A =
        1     2     3     4     5
        2     4     6     8    10
        3     6     9    12    15
```

Ejercicio de práctica 4.2

1. El área de un rectángulo es longitud por ancho (área = longitud × ancho). Encuentre las áreas de los rectángulos con longitudes de 1, 3 y 5 cm y con anchos de 2, 4, 6 y 8 cm. (Debe tener 12 respuestas.)
2. El volumen de un cilindro es volumen = $\pi r^2 h$. Encuentre el volumen de los contenedores cilíndricos con radios desde 0 hasta 12 m y alturas desde 10 hasta 20 m. Aumente la dimensión del radio por 3 metros y la altura por 2 metros conforme abarca los dos rangos.



EJEMPLO 4.2

Distancia al horizonte

Probablemente ha experimentado estar de pie en lo alto de una colina o montaña y sentido que puede ver hasta el infinito. ¿Realmente cuán lejos puede ver? Depende de la altura de la montaña y del radio de la Tierra, como se muestra en la figura 4.2. La distancia hasta el horizonte es muy diferente en la Luna que en la Tierra, porque el radio es diferente para cada una.

Con el teorema de Pitágoras se ve que

$$R^2 + d^2 = (R + h)^2$$

y despejar d produce $d = \sqrt{h^2 + 2Rh}$.

A partir de esta última expresión, encuentre la distancia hasta el horizonte en la Tierra y en la Luna, para montañas desde 0 hasta 8000 metros. (El monte Everest tiene 8850 metros de alto.) El radio de la Tierra es 6378 km y el de la Luna es de 1737 km.

1. Establezca el problema.
   Encontrar la distancia hasta el horizonte desde lo alto de una montaña en la Luna y en la Tierra.

2. Describa las entradas y salidas.

   Entrada

Radio de la Luna	1737 km
Radio de la Tierra	6378 km
Altura de las montañas	0 a 8000 metros

   Salida

   Distancia hasta el horizonte, en kilómetros

3. Desarrolle un ejemplo a mano.

$$d = \sqrt{h^2 + 2Rh}$$

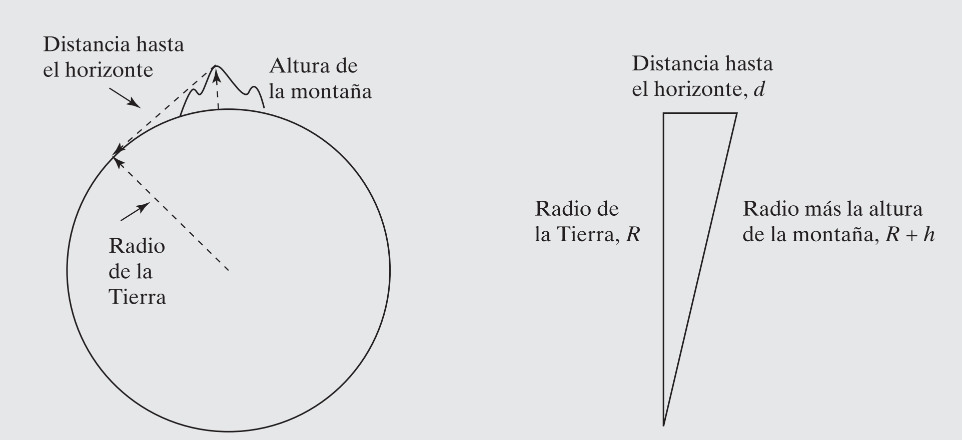


Figura 4.2
Distancia hasta el horizonte.

Con el radio de la Tierra y una montaña de 8000 metros se obtiene

$$d = \sqrt{(8\text{ km})^2 + 2 \times 6378\text{ km} \times 8\text{ km}} = 319\text{ km}$$

4. Desarrolle una solución MATLAB.

```
%Ejemplo 4.2
%Encontrar la distancia hasta el horizonte
%Definir la altura de las montañas
% en metros
clear, clc
format bank
%Definir el vector altura
h=0:1000:8000;
%Convierte metros a km
h=h/1000;
%Definir los radios de la Luna y la Tierra
radius = [1737      6378];
%Mapea los radios y alturas en una retícula 2-D
[Radius,H]=meshgrid(radius,h);
%Calcula la distancia hasta el horizonte
d=sqrt(H.^2 + 2*H.*Radius)
```

Ejecutar el archivo-m precedente regresa una tabla de las distancias hasta el horizonte tanto en la Luna como en la Tierra:

```
d =
           0            0
       58.95       112.95
       83.38       159.74
      102.13       195.65
      117.95       225.92
      131.89       252.60
      144.50       276.72
      156.10       298.90
      166.90       319.55
```

5. Ponga a prueba la solución.
   Compare la solución MATLAB con la solución a mano. La distancia hasta el horizonte desde cerca de la cima del monte Everest (8000 m) es de más de 300 km e iguala el valor calculado en MATLAB.

EJEMPLO 4.3

Caída libre

La ecuación general para la distancia que recorre un cuerpo en caída libre (sin tomar en cuenta la fricción del aire) es

$$d = \frac{1}{2}gt^2$$

donde

d = distancia,
g = aceleración debida a la gravedad, y
t = tiempo.

Cuando un satélite orbita un planeta, está en caída libre. Muchas personas creen que, cuando el transbordador espacial entra en órbita, deja detrás la gravedad; pero la gravedad es lo que mantiene al transbordador en órbita. El transbordador (o cualquier satélite) en realidad cae hacia la Tierra (figura 4.3). Si va lo suficientemente rápido de manera horizontal, permanece en órbita; si va muy lentamente, golpea el suelo.

El valor de la constante g, la aceleración debida a la gravedad, depende de la masa del planeta. En diferentes planetas, g tiene diferentes valores (tabla 4.2).

Encuentre qué tan lejos caería un objeto en tiempos desde 0 hasta 100 segundos en cada uno de los planetas del sistema solar y en la Luna.

1. Establezca el problema.
   Encontrar la distancia que recorre un objeto en caída libre en planetas con diferentes gravedades.

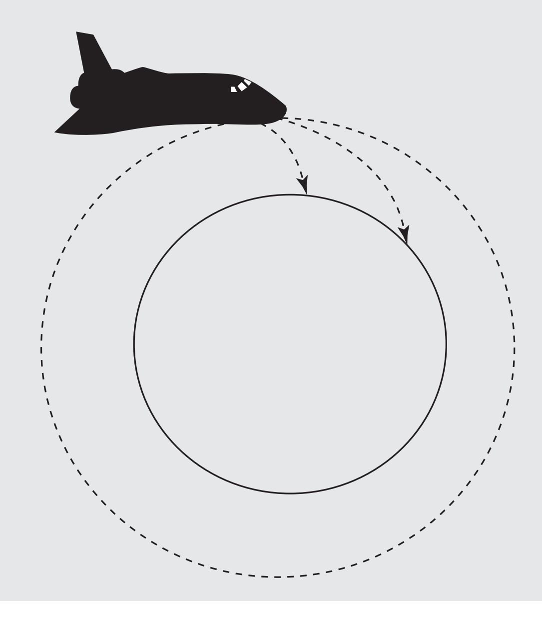
2. Describa las entradas y salidas

   Entrada Valor de g, la aceleración debida a la gravedad, en cada uno de los planetas y la Luna

   Tiempo = 0 a 100 s

   Salida Distancias calculadas para cada planeta y la Luna

3. Desarrolle un ejemplo a mano.
   $d = ½\, gt^2$, así que en Mercurio, a 100 segundos:
   $d = ½ \times 3.7\ \text{m/s}^2 \times 100^2\ \text{s}^2$
   $d = 18{,}500$ m



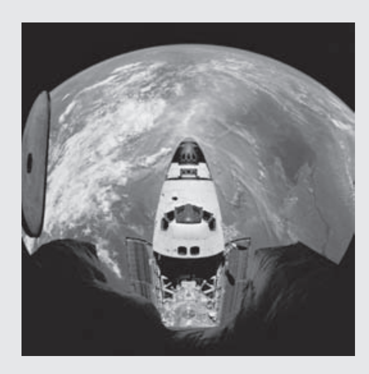


Figura 4.3
El transbordador espacial cae constantemente hacia la Tierra. (Cortesía de NASA/Jet Propulsion Laboratory.)

Tabla 4.2 Aceleración debida a la gravedad en el sistema solar

Mercurio	$g = 3.7\ \mathrm{m/s^2}$
Venus	$g = 8.87\ \mathrm{m/s^2}$
Tierra	$g = 9.8\ \mathrm{m/s^2}$
Luna	$g = 1.6\ \mathrm{m/s^2}$
Marte	$g = 3.7\ \mathrm{m/s^2}$
Júpiter	$g = 23.12\ \mathrm{m/s^2}$
Saturno	$g = 8.96\ \mathrm{m/s^2}$
Urano	$g = 8.69\ \mathrm{m/s^2}$
Neptuno	$g = 11.0\ \mathrm{m/s^2}$
Plutón	$g = .58\ \mathrm{m/s^2}$

4. Desarrolle una solución MATLAB.

```
%Ejemplo 4.3
%Caída libre
clear, clc
%Intente el problema primero sólo con dos planetas y una
%  retícula burda

format bank
%Definir constantes para aceleración debida a gravedad en
%Mercurio y la Luna de la Tierra
G = [3.7, 8.87];
T=0:10:100;  %Definir vector tiempo

%Mapea G y T en matrices 2D
[g,t]=meshgrid(G,T);
%Calcula las distancias
d=1/2*g.*t.^2
```

La ejecución del archivo-m precedente regresa los siguientes valores de la distancia que se recorre en Mercurio y en la Luna de la Tierra.

```
d =
             0            0
        185.00       443.50
        740.00      1774.00
       1665.00      3991.50
       2960.00      7096.00
       4625.00     11087.50
       6660.00     15966.00
       9065.00     21731.50
      11840.00     28384.00
      14985.00     35923.50
      18500.00     44350.00
```

5. Ponga a prueba la solución.
   Compare la solución MATLAB con la solución a mano. Se puede ver que la distancia que se recorre en Mercurio a 100 segundos es 18,500 m, que corresponde al cálculo a mano.

   El archivo-m incluye los cálculos sólo para los dos primeros planetas y se realizó primero para trabajar cualesquiera dificultades de programación. Una vez que se confirma que el programa funciona, es fácil volver a hacerlo con los datos para todos los planetas:

```
%Vuelva a hacer el problema con todos los datos
clear, clc
format bank
%Defina constantes
```

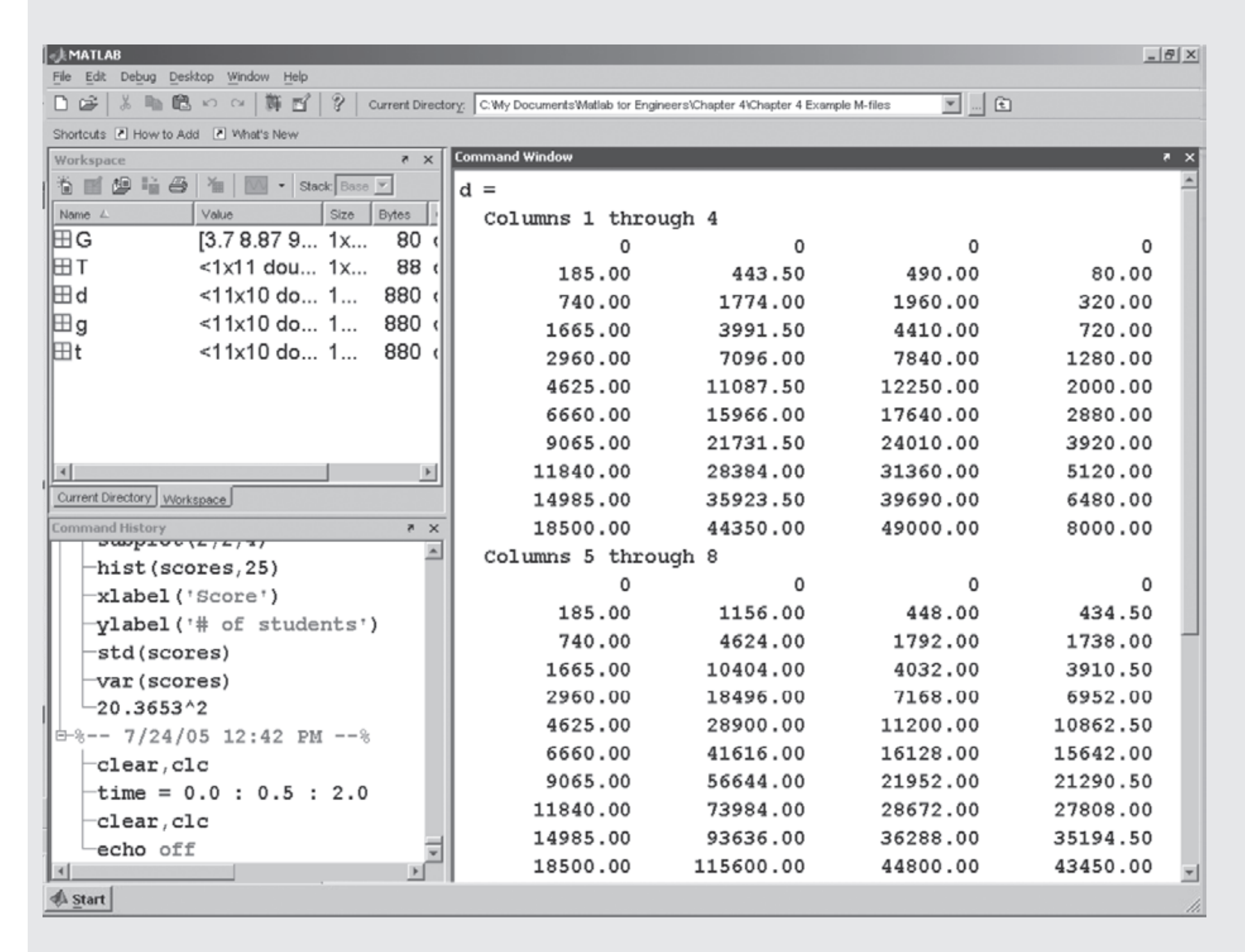


Figura 4.4
Resultados de los cálculos de distancia para un objeto que cae en cada uno de los planetas.

```
G = [3.7, 8.87, 9.8, 1.6, 3.7, 23.12 8.96, 8.69, 11.0, 0.58];
T=0:10:100;
%Mapea G y T en matrices 2D
[g,t]=meshgrid(G,T);
%Calcula las distancias
d=1/2*g.*t.^2
```

Existen muchas cosas importantes a notar acerca de los resultados que se muestran en la figura 4.4. Primero, observe la ventana del área de trabajo. **G** es una matriz 1×10 (un valor para cada uno de los planetas y la Luna) y **T** es una matriz 1×11 (11 valores de tiempo). Sin embargo, tanto **g** como **t** son matrices 11×10, el resultado de la operación **meshgrid**. Los resultados que se muestran en la ventana de comandos se formatearon con el comando **format bank** para hacer la salida más fácil de leer; de otro modo, habría habido un factor de escala común.

Sugerencia

Mientras crea un programa MATLAB en la ventana de edición, tal vez quiera comentar aquellas partes del código que sabe que sí funcionan y quitar los comentarios más tarde. Aunque puede hacer esto al agregar un % a la vez en cada línea, es más fácil seleccionar **text** de la barra de menú. Sólo resalte la parte del código que quiere comentar y luego elija **comment** del menú desplegable **text**. Para borrar los comentarios, resalte y seleccione **uncomment** del menú desplegable **text** (text → uncomment). También puede acceder a este menú haciendo clic en el lado derecho de la ventana de edición.

4.3 MATRICES ESPECIALES

MATLAB contiene un grupo de funciones que generan matrices especiales; en la tabla 4.3 se presentan algunas de dichas funciones.

4.3.1 Matriz de ceros

A veces es útil crear una matriz de ceros. Cuando se usa la función **zeros** con un solo argumento escalar de entrada, se genera una matriz cuadrada:

```
A = zeros(3)
A =
        0       0       0
        0       0       0
        0       0       0
```

Tabla 4.3 Funciones para crear y manipular matrices

`zeros(m)`	Crea una matriz $m \times m$ de ceros	`zeros(3)`
		`ans =`
		`0 0 0`
		`0 0 0`
		`0 0 0`

`zeros(m,n)`	Crea una matriz $m \times n$ de ceros	`zeros(2,3)`
		`ans =`
		`0 0 0`
		`0 0 0`

`ones(m)`	Crea una matriz $m \times m$ de unos	`ones(3)`
		`ans =`
		`1 1 1`
		`1 1 1`
		`1 1 1`

`ones(m,n)`	Crea una matriz $m \times n$ de unos	`ones(2,3)`
		`ans =`
		`1 1 1`
		`1 1 1`

`diag(A)`	Extrae la diagonal de una matriz bidimensional **A**	`A=[1 2 3; 3 4 5; 1 2 3];`
		`diag(A)`
		`ans =`
		`1`
		`4`
		`3`

	Para cualquier vector **A**, crea una matriz cuadrada con **A** como la diagonal. Verifique la función **help** para otras formas en que se puede usar la función **diag**	`A=[1 2 3];`
		`diag(A)`
		`ans =`
		`1 0 0`
		`0 2 0`
		`0 0 3`

`fliplr`	Voltea una matriz en su imagen especular de derecha a izquierda	`A=[1 0 0; 0 2 0; 0 0 3];`
		`fliplr(A)`
		`ans =`
		`0 0 1`
		`0 2 0`
		`3 0 0`

`flipud`	Voltea una matriz verticalmente	`flipud(A)`
		`ans =`
		`0 0 3`
		`0 2 0`
		`1 0 0`

`magic(m)`	Crea una matriz "mágica" $m \times m$	`magic(3)`
		`ans =`
		`8 1 6`
		`3 5 7`
		`4 9 2`

Si se usan dos argumentos escalares, el primer valor especifica el número de filas y el segundo argumento especifica el número de columnas:

```
B = zeros(3,2)
B =
        0     0
        0     0
        0     0
```

Idea clave: use una matriz de ceros o unos como un marcador de posición para futuros cálculos.

4.3.2 Matriz de unos

La función **ones** es similar a la función **zeros**, pero crea una matriz de unos:

```
A = ones(3)
A =
        1     1     1
        1     1     1
        1     1     1
```

Como con la función **zeros**, si se usan dos entradas, se puede controlar el número de filas y columnas:

```
B = ones(3,2)
B =
        1     1
        1     1
        1     1
```

Las funciones **zeros** y **ones** son útiles para crear matrices con valores "marcadores de posición" (placeholders) que se llenarán más tarde. Por ejemplo, si quiere un vector de cinco números, los cuales sean iguales a π, primero puede crear un vector de unos:

```
a=ones(1,5)
```

Esto da

```
a =
      1     1     1     1     1
```

Luego multiplicar por π:

```
b=a*pi
```

El resultado es

```
b =
     3.1416    3.1416    3.1416    3.1416    3.1416
```

El mismo resultado se podría obtener al sumar π a una matriz de ceros. Por ejemplo,

```
a=zeros(1,5);
b=a+pi
```

produce

```
b =
     3.1416    3.1416    3.1416    3.1416    3.1416
```

Una matriz de marcadores de posición es especialmente útil en programas MATLAB con una estructura de bucle (loop), porque puede reducir el tiempo que tarda en ejecutarse el bucle.

4.3.3 Matrices diagonal

Puede usar la función **diag** para extraer la diagonal de una matriz. Por ejemplo, si se define una matriz cuadrada

```
A=[1 2 3; 3 4 5; 1 2 3];
```

entonces usar la función

```
diag(A)
```

extrae la diagonal principal y produce los siguientes resultados:

```
ans =
    1.00
    4.00
    3.00
```

Se pueden extraer otras diagonales al definir una segunda entrada, **k**, a **diag**. Los valores positivos de **k** especifican diagonales en la esquina superior derecha de la matriz, y los valores negativos especifican diagonales en la esquina inferior izquierda de la matriz. (Véase la figura 4.5.)

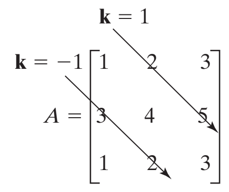


Figura 4.5
Cada diagonal en una matriz se puede describir mediante el parámetro **k**.

```
diag(A,1)
```

regresa

```
ans =
    2
    5
```

Si, en lugar de usar una matriz bidimensional como entrada para la función **diag**, se usa un vector como

```
B=[1 2 3];
```

entonces MATLAB usa el vector para los valores a lo largo de la diagonal de una nueva matriz y llena los elementos restantes con ceros:

```
diag(B)
ans =
     1     0     0
     0     2     0
     0     0     3
```

Al especificar un segundo parámetro, se puede mover la diagonal a cualquier lugar en la matriz:

```
diag(B,1)
ans =
     0     1     0     0
     0     0     2     0
     0     0     0     3
     0     0     0     0
```

4.3.4 Matrices mágicas

MATLAB incluye una función matricial llamada **magic** que genera una matriz con propiedades inusuales. Parece no haber algún uso práctico para las matrices mágicas, excepto que son divertidas. En una matriz mágica, la suma de todas las columnas es la misma, al igual que la suma de todas las filas. Un ejemplo es

```
A=magic(4)
A =
   16     2     3    13
    5    11    10     8
    9     7     6    12
    4    14    15     1

sum(A)
ans =
    34    34    34    34
```

Para encontrar la suma de las filas, es necesario trasponer la matriz:

```
sum(A')
ans =
  34    34    34    34
```

No sólo la suma de todas las columnas y filas es la misma, también la suma de las diagonales es la misma. La diagonal de izquierda a derecha es

```
diag(A)
ans =
  16
  11
   6
   1
```

La suma de la diagonal es el mismo número que la suma de las filas y columnas:

```
sum(diag(A))
ans =
  34
```

Finalmente, para encontrar la diagonal de inferior izquierda a superior derecha, primero se tiene que "voltear" la matriz y luego encontrar la suma de la diagonal:

```
fliplr(A)
ans =
  13     3     2    16
   8    10    11     5
  12     6     7     9
   1    15    14     4

diag(ans)
ans =
  13
  10
```



Figura 4.6
"Melancolía", de Albrecht Dürer, 1514.

```
    7
    4

 sum(ans)
 ans =
   34
```

En la figura 4.6 se muestra uno de los primeros ejemplos documentados de un cuadrado mágico, en el grabado en madera "Melancolía", creado por Albrecht Dürer en 1514. Los expertos creen que el cuadrado era una referencia a los conceptos alquímicos populares en la época. La fecha de un grabado se incluye en los dos cuadrados medios de la fila inferior. (Véase la figura 4.7.)

Los cuadrados mágicos han fascinado durante siglos a los matemáticos tanto profesionales como aficionados. Por ejemplo, Benjamin Franklin experimentó con los cuadrados mágicos. Usted puede crear cuadrados mágicos de cualquier tamaño mayores que 2 × 2 en MATLAB. Sin embargo, son posibles otros cuadrados mágicos; la solución de MATLAB no es la única.

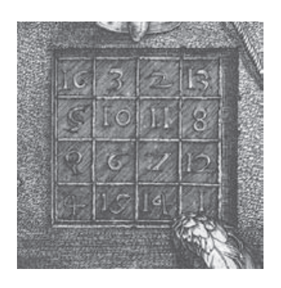


Figura 4.7
Albrecht Dürer incluyó la fecha del grabado (1514) en el cuadrado mágico.

Ejercicio de práctica 4.3

1. Cree una matriz 3 × 3 de ceros.
2. Cree una matriz 3 × 4 de ceros.
3. Cree una matriz 3 × 3 de unos.
4. Cree una matriz 5 × 3 de unos.
5. Cree una matriz 4 × 6 en la que todos los elementos tengan un valor de pi.
6. Use la función **diag** para crear una matriz cuya diagonal tenga valores de 1, 2, 3.
7. Cree una matriz mágica 10 × 10.
   a. Extraiga la diagonal de esta matriz.
   b. Extraiga la diagonal que corre de inferior izquierda a superior derecha de esta matriz.
   c. Confirme que la suma de las filas, columnas y diagonales son todas iguales.

RESUMEN

Este capítulo se concentró en la manipulación de matrices, una capacidad que permite al usuario crear matrices complicadas al combinar unas más pequeñas. También le permite extraer porciones de una matriz existente. El operador dos puntos es especialmente útil para dichas operaciones. El operador dos puntos se debe interpretar como "todas las filas" o "todas las columnas" cuando se usa en lugar de una designación de fila o columna. Se debe interpretar como "desde _ hasta _" cuando se usa entre números de fila o columna. Por ejemplo,

```
A(:,2:3)
```

se debe interpretar como "todas las filas en la matriz **A** y todas las columnas desde 2 hasta 3". Cuando se usa solo como el índice exclusivo, como en **A(:)**, crea una matriz que es una sola columna a partir de una representación bidimensional. En realidad la computadora almacena toda la información de arreglo como una lista, lo que hace tanto de la notación de índice solo como de la notación fila-columna alternativas útiles para especificar la ubicación de un valor en una matriz.

La función **meshgrid** es extremadamente útil, dado que se puede usar para mapear vectores en matrices bidimensionales, lo que hace posible la realización de cálculos de arreglo con vectores de tamaño desigual.

MATLAB contiene algunas funciones que facilitan la creación de matrices especiales:

- **zeros**, que se usa para crear una matriz compuesta completamente de ceros.
- **ones**, que se usa para crear una matriz compuesta completamente de unos.
- **diag**, que se puede usar para extraer la diagonal de una matriz o, si la entrada es un vector, se puede usar para crear una matriz cuadrada.
- **magic**, que se puede usar para crear una matriz con la propiedad inusual de que todas las filas y columnas suman el mismo valor, así como las diagonales.

Además, se incluyeron algunas funciones que permiten al usuario "voltear" la matriz de izquierda a derecha o de arriba abajo.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona y describe brevemente todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres especiales	
`:`	operador dos puntos
`...`	elipsis, que indica continuación en la línea siguiente
`[]`	matriz vacía

Comandos y funciones	
`meshgrid`	mapea vectores en un arreglo bidimensional
`zeros`	crea una matriz de ceros
`ones`	crea una matriz de unos
`diag`	extrae la diagonal de una matriz
`fliplr`	voltea una matriz en su imagen especular, de izquierda a derecha
`flipud`	voltea una matriz verticalmente
`magic`	crea una matriz "mágica"

TÉRMINOS CLAVE

elementos
números índice
matrices mágicas
mapeo
subíndices

PROBLEMAS

Manipulación de matrices

4.1 Cree las siguientes matrices y úselas en los ejercicios que siguen:

$$a = \begin{bmatrix} 15 & 3 & 22 \\ 3 & 8 & 5 \\ 14 & 3 & 82 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 5 \\ 6 \end{bmatrix} \qquad c = [12 \quad 18 \quad 5 \quad 2]$$

(a) Cree una matriz llamada **d** a partir de la tercera columna de la matriz **a**.
(b) Combine la matriz **b** y la matriz **d** para crear la matriz **e**, una matriz bidimensional con tres filas y dos columnas.
(c) Combine la matriz **b** y la matriz **d** para crear la matriz **f**, una matriz unidimensional con seis filas y una columna.
(d) Cree una matriz **g** a partir de la matriz **a** y los primeros tres elementos de la matriz **c**, con cuatro filas y tres columnas.
(e) Cree una matriz **h** con el primer elemento igual a $a_{1,3}$, el segundo elemento igual a $c_{1,2}$ y el tercer elemento igual a $b_{2,1}$.

4.2 Cargue el archivo **thermo_scores.dat** proporcionado por su instructor, o ingrese la matriz de la parte superior de la página 130 y llámela **thermo_scores**. (Sólo ingrese los números.)

(a) Extraiga las calificaciones y número de estudiante para el estudiante 5 en un vector fila llamado **student_5**.
(b) Extraiga las calificaciones para el examen 1 en un vector columna llamado **test_1**.
(c) Encuentre la desviación estándar y la varianza para cada examen.
(d) Si supone que cada examen valía 100 puntos, encuentre la calificación final total y el porcentaje final de cada estudiante. (Tenga cuidado de no sumar el número de estudiante.)
(e) Cree una tabla que incluya los porcentajes finales y las calificaciones de la tabla original.

Estudiante número	Examen 1	Examen 2	Examen 3
1	68	45	92
2	83	54	93
3	61	67	91
4	70	66	92
5	75	68	96
6	82	67	90
7	57	65	89
8	5	69	89
9	76	62	97
10	85	52	94
11	62	34	87
12	71	45	85
13	96	56	45
14	78	65	87
15	76	43	97
16	68	76	95
17	72	65	89
18	75	67	88
19	83	68	91
20	93	90	92

(f) Ordene la matriz sobre la base del porcentaje final, de mayor a menor (en orden descendente), y conserve juntos los datos de cada fila. (Es posible que necesite consultar la función **help** para determinar la sintaxis adecuada.)

4.3 Considere la siguiente tabla:

Tiempo (h)	Termocople 1 °F	Termocople 2 °F	Termocople 3 °F
0	84.3	90.0	86.7
2	86.4	89.5	87.6
4	85.2	88.6	88.3
6	87.1	88.9	85.3
8	83.5	88.9	80.3
10	84.8	90.4	82.4
12	85.0	89.3	83.4
14	85.3	89.5	85.4
16	85.3	88.9	86.3
18	85.2	89.1	85.3
20	82.3	89.5	89.0
22	84.7	89.4	87.3
24	83.6	89.8	87.2

(a) Cree un vector columna llamado **times** que vaya de 0 a 24 en incrementos de 2 horas.

(b) Su instructor le puede proporcionar las temperaturas de termocople en un archivo llamado **thermocouple.dat**, o tal vez necesite crear usted mismo una matriz llamada **thermocouple** escribiéndole los datos.

(c) Combine el vector **times** que creó en la parte (a) con los datos de **thermocouple** para crear una matriz que corresponda a la tabla de este problema.

(d) Recuerde que las funciones **max** y **min** pueden regresar no sólo los valores máximos en una columna, sino también el número de elemento donde ocurren dichos valores. Use esta capacidad para determinar los valores de **times** a los que ocurren los máximos y mínimos en cada columna.

4.4 Suponga que un archivo llamado **sensor.dat** contiene información recopilada de un conjunto de sensores. Su instructor le puede proporcionar este archivo, o usted puede ingresarlo a mano a partir de los siguientes datos:

Tiempo (s)	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5
0.0000	70.6432	68.3470	72.3469	67.6751	73.1764
1.0000	73.2823	65.7819	65.4822	71.8548	66.9929
2.0000	64.1609	72.4888	70.1794	73.6414	72.7559
3.0000	67.6970	77.4425	66.8623	80.5608	64.5008
4.0000	68.6878	67.2676	72.6770	63.2135	70.4300
5.0000	63.9342	65.7662	2.7644	64.8869	59.9772
6.0000	63.4028	68.7683	68.9815	75.1892	67.5346
7.0000	74.6561	73.3151	59.7284	68.0510	72.3102
8.0000	70.0562	65.7290	70.6628	63.0937	68.3950
9.0000	66.7743	63.9934	77.9647	71.5777	76.1828
10.0000	74.0286	69.4007	75.0921	77.7662	66.8436
11.0000	71.1581	69.6735	62.0980	73.5395	58.3739
12.0000	65.0512	72.4265	69.6067	79.7869	63.8418
13.0000	76.6979	67.0225	66.5917	72.5227	75.2782
14.0000	71.4475	69.2517	64.8772	79.3226	69.4339
15.0000	77.3946	67.8262	63.8282	68.3009	71.8961
16.0000	75.6901	69.6033	71.4440	64.3011	74.7210
17.0000	66.5793	77.6758	67.8535	68.9444	59.3979
18.0000	63.5403	66.9676	70.2790	75.9512	66.7766
19.0000	69.6354	63.2632	68.1606	64.4190	66.4785

Cada fila contiene un conjunto de lecturas de sensor, donde la primera fila contiene valores recopilados a 0 segundos, la segunda fila contiene valores recopilados a 1.0 segundos, etcétera.

(a) Lea el archivo de datos e imprima el número de sensores y el número de segundos de datos contenidos en el archivo. (*Sugerencia:* Use la función **size**, no sólo cuente los dos números.)
(b) Encuentre los valores máximo y mínimo registrados en cada sensor. Use MATLAB para determinar en qué momentos ocurrieron.
(c) Encuentre la media y la desviación estándar para cada sensor y para todos los valores de datos recopilados. Recuerde: la columna 1 no contiene datos de sensor; contiene datos de tiempo.

Problemas con dos variables

4.5 El área de un triángulo es área = ½ base × altura. (Véase la figura P4.5.) Encuentre el área de un grupo de triángulos cuya base varía de 0 a 10 metros y cuya altura varía de 2 a 6 metros. Elija un espaciamiento adecuado para sus variables de cálculo. Su respuesta debe ser una matriz bidimensional.

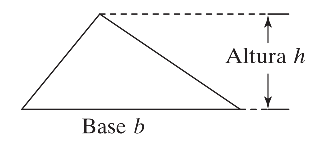


Figura P4.5
El área de un triángulo.

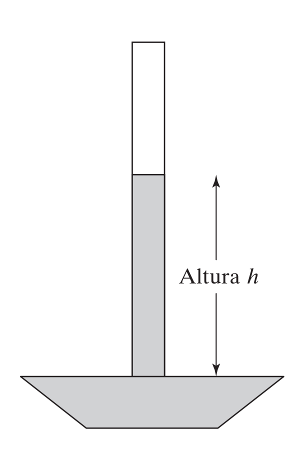


Figura P4.6
Barómetro.

4.6 Un barómetro (véase la figura P4.6) se usa para medir la presión atmosférica y se llena con un fluido de alta densidad. En el pasado se usaba mercurio, pero desde entonces se sustituyó con una diversidad de otros fluidos debido a sus propiedades tóxicas. La presión p medida por un barómetro es la altura de la columna de fluido, h, por la densidad del líquido, ρ, por la aceleración debida a la gravedad, g, o

$$P = h\rho g$$

Esta ecuación se puede despejar para la altura:

$$h = \frac{P}{\rho g}$$

Encuentre la altura a la que la columna de líquido se elevará para presiones desde 0 hasta 10 kPa para dos barómetros diferentes. Suponga que el primero usa mercurio, con una densidad de 13.56 g/cm^3 (13,560 kg/m^3) y que el segundo usa agua, con una densidad de 1.0 g/cm^3 (1000 kg/m^3). La aceleración debida a la gravedad es 9.81 m/s^2. Antes de comenzar a calcular, asegúrese de verificar las unidades. La medida métrica de la presión es un Pascal (Pa), igual a 1 kg m/s^2. Un kPa es 1000 veces mayor que un Pa. Su respuesta debe ser una matriz bidimensional.

4.7 La ley del gas ideal $Pv = RT$ describe el comportamiento de muchos gases. Cuando se despeja v (el volumen específico, m^3/kg) la ecuación se puede escribir

$$v = \frac{RT}{P}$$

Encuentre el volumen específico para el aire, para temperaturas de 100 a 1000 K y para presiones de 100 kPa a 1000 kPa. El valor de R para el aire es 0.2870 kJ/(kg K). En esta formulación de la ley del gas ideal, R es diferente para cada gas. Existen otras formulaciones en las que R es una constante y el peso molecular del gas se debe incluir en el cálculo. Aprenderá más acerca de esta ecuación en las clases de química y termodinámica. Su respuesta debe ser una matriz bidimensional.

Matrices especiales

4.8 Cree una matriz de ceros del mismo tamaño que las matrices **a**, **b** y **c** del problema 4.1. (Use la función **size** para ayudarse a cumplir esta tarea.)

4.9 Cree una matriz mágica de 6×6.

(a) ¿Cuál es la suma de cada una de las filas?
(b) ¿Cuál es la suma de cada una de las columnas?
(c) ¿Cuál es la suma de cada una de las diagonales?

4.10 Extraiga una matriz 3×3 de la esquina superior izquierda de la matriz mágica que creó en el problema 4.9. ¿Ésta también es una matriz mágica?

4.11 Cree una matriz mágica de 5 × 5 llamada **a**.

(a) **a** por una constante, como 2, ¿también es una matriz mágica?
(b) Si eleva al cuadrado cada elemento de **a**, ¿la nueva matriz es una matriz mágica?
(c) Si suma una constante a cada elemento, ¿la nueva matriz es una matriz mágica?
(d) Cree una matriz 10 × 10 a partir de los siguientes componentes (véase la figura P4.11):

- la matriz **a**.
- 2 por la matriz **a**.
- una matriz formada por elevar al cuadrado cada elemento de **a**.
- 2 más la matriz **a**.

¿Su resultado es una matriz mágica? ¿La forma en la que ordena los componentes afecta su respuesta?

a	**2*a**
a^2	**a+2**

Figura P4.11
Cree una matriz a partir de otras matrices.



CAPÍTULO

5

Graficación

Objetivos

Después de leer este capítulo, el alumno será capaz de

- crear y etiquetar gráficas bidimensionales.
- ajustar la apariencia de sus gráficas.
- dividir la ventana de graficación en subgráficas.
- crear gráficas tridimensionales.
- usar las herramientas de graficación interactivas de MATLAB.

INTRODUCCIÓN

Las tablas de datos muy grandes son difíciles de interpretar. Los ingenieros usan técnicas de graficación para hacer que la información se entiendan fácilmente. Con una gráfica es fácil identificar tendencias, elegir altos y bajos y aislar puntos de datos que pueden ser mediciones o cálculos de errores. Las gráficas también se pueden usar como una rápida verificación para determinar si una solución de computadora produce los resultados esperados.

5.1 GRÁFICAS BIDIMENSIONALES

La gráfica más útil para los ingenieros es la gráfica *x*-*y*. Un conjunto de pares ordenados se usa para identificar puntos sobre una gráfica bidimensional; luego los puntos se conectan con líneas rectas. Los valores de *x* y *y* se pueden medir o calcular. Por lo general, a la variable independiente se le da el nombre *x* y se grafica en el eje *x*, y la variable dependiente recibe el nombre *y* y se grafica en el eje *y*.

5.1.1 Graficación básica

Gráficas x-y simples

Una vez definidos los vectores de valores *x* y valores *y*, MATLAB hace fácil la creación de gráficas. Suponga, a través de medición, se obtiene un conjunto de datos tiempo contra distancia.

Se pueden almacenar los valores de tiempo en un vector llamado **x** (el usuario puede definir cualquier nombre conveniente) y los valores de distancia en un vector llamado **y**:

```
x = [0:2:18];
y = [0, 0.33, 4.13, 6.29, 6.85, 11.19, 13.19, 13.96,
     16.33, 18.17];
```

Para graficar estos puntos, use el comando **plot**, con **x** y **y** como argumentos:

```
plot(x,y)
```

Idea clave: incluya siempre unidades en las etiquetas de los ejes.

Automáticamente se abre una ventana de gráficas, la que MATLAB llama figure 1. En la figura 5.1 se muestra la gráfica resultante. (Pueden ocurrir ligeras variaciones en la escala, dependiendo del tipo de computadora y tamaño de la ventana de gráficas.)

Títulos, etiquetas y retículas

La buena práctica ingenieril requiere que se incluyan unidades y un título en las gráficas. Los siguientes comandos agregan un título, etiquetas a los ejes *x* y *y*, y una retícula de fondo:

```
plot(x,y)
title('Experimento de laboratorio 1')
xlabel('Tiempo, seg' )
ylabel('Distancia, pies')
grid on
```

Estos comandos generan la gráfica de la figura 5.2. También se pueden escribir en una sola línea o dos, separadas por comas:

```
plot(x,y) , title('Experimento de laboratorio 1') ,
xlabel('Tiempo, seg.' ), ylabel('Distancia, pies'), grid
```

cadena: lista de caracteres encerrados por apóstrofes

Conforme escribe los comandos anteriores en MATLAB, note que el color del texto cambia a rojo cuando ingresa un apóstrofe ('). Esto le advierte que comienza una cadena (string). El color cambia a púrpura cuando escribe el apóstrofe final ('), lo que indica que completó la cadena. Poner atención a estos auxiliares visuales le ayudará a evitar errores de codificación. MATLAB 6 usa diferentes pistas de color, pero la idea es la misma.

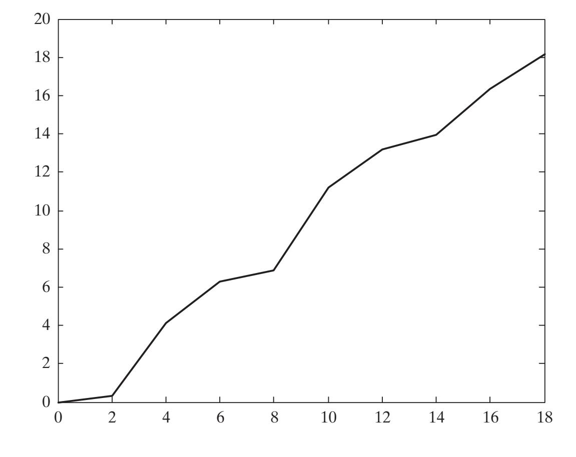


Figura 5.1
Gráfica simple de tiempo contra distancia creada en MATLAB.

Tiempo, seg	Distancia, pies
0	0
2	0.33
4	4.13
6	6.29
8	6.85
10	11.19
12	13.19
14	13.96
16	16.33
18	18.17

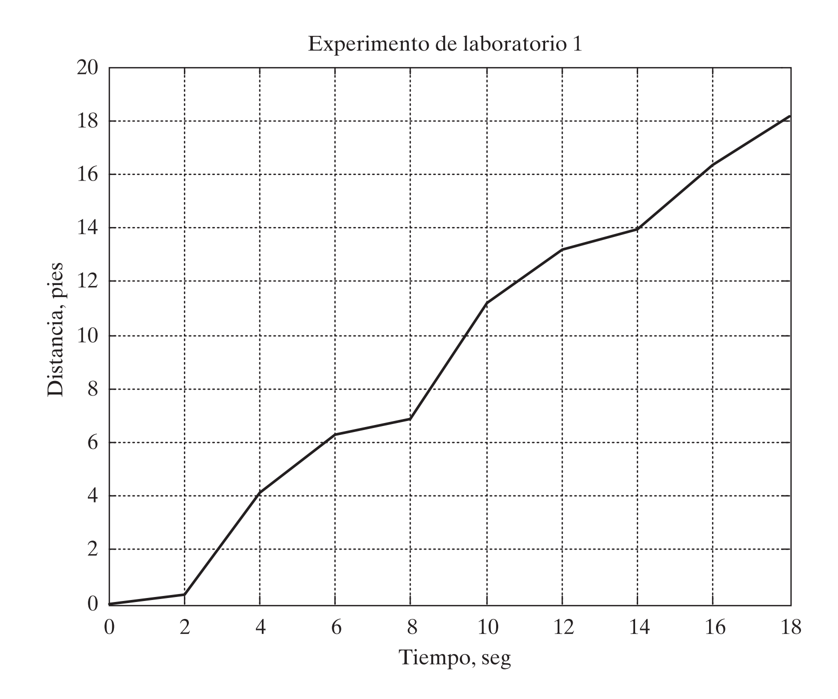


Figura 5.2
Agregar una retícula, un título y etiquetas hacen una gráfica más fácil de interpretar.

Si trabaja en la ventana de comandos, la ventana de gráficas se abrirá encima de las otras ventanas. (Véase la figura 5.3.) Para continuar trabajando, haga clic en la ventana de comandos o minimice la ventana de gráficas. También puede redimensionar la ventana de gráficas a cualquier tamaño que sea conveniente o agregarla al escritorio de MATLAB al seleccionar la flecha de acoplamiento (docking) abajo del icono exit en la esquina superior derecha de la ventana de la figura.

Sugerencia

Una vez que hace clic en la ventana de comandos, la ventana de figura se oculta detrás de la ventana actual. Para ver los cambios a su figura, necesitará seleccionar la figura de la barra de tareas de Windows en la parte inferior de la pantalla.

Sugerencia

Debe crear una gráfica *antes* de agregarle título y etiquetas. Si primero especifica el título y las etiquetas, se borrarán cuando ejecute el comando plot.

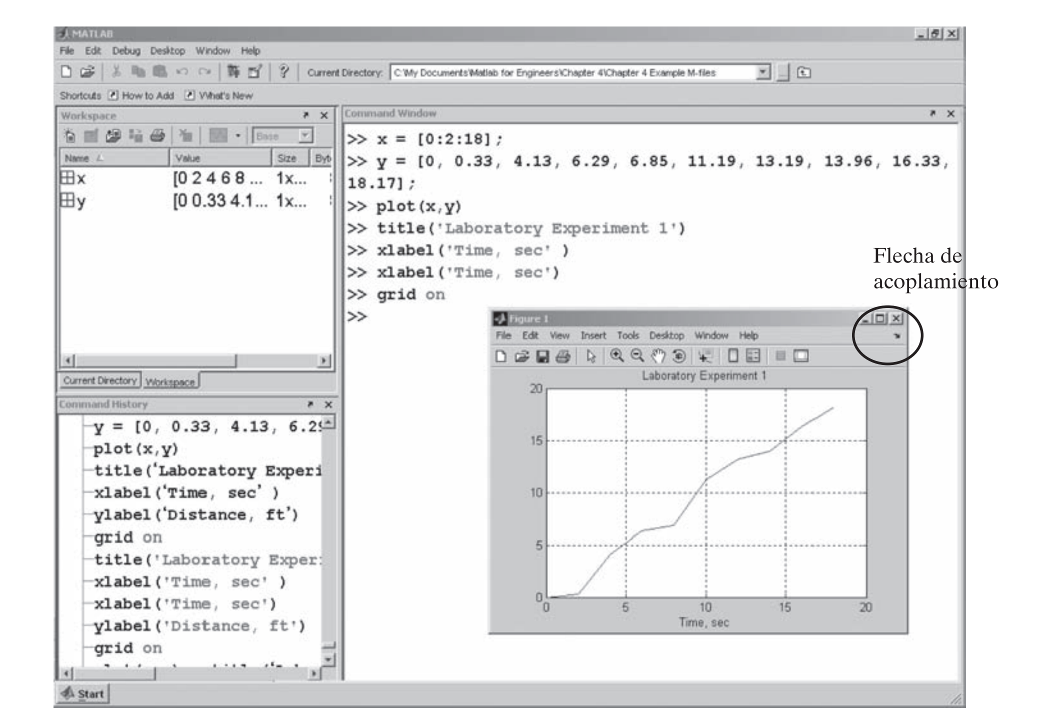


Figura 5.3
La ventana de gráficas se abre arriba de la ventana de comandos. Puede redimensionarla a una forma conveniente.

> **Sugerencia**
>
> Puesto que se usa un apóstrofe al final de la cadena que se ingresa en los comandos **xlabel**, **ylabel** y **title**, MATLAB interpreta el apóstrofe como el final de la cadena. Ingresar doble apóstrofe, con en **xlabel('Holly''s Data')**, le permitirá usar apóstrofes en su texto.

Creación de gráficas múltiples

Si trabaja en un archivo-m cuando solicita una gráfica, y luego continúa con más cálculos, MATLAB generará y desplegará la ventana de gráficas y luego regresará inmediatamente a ejecutar el resto de los comandos en el programa. Si solicita una segunda gráfica, la gráfica que creó se sobrescribirá. Existen dos posibles soluciones a este problema: use el comando **pause** para detener temporalmente la ejecución de su programa archivo-m, o cree una segunda figura, con el uso de la función **figure**.

El comando **pause** detiene la ejecución del programa hasta que se oprime alguna tecla. Si quiere pausar durante un número específico de segundos, use el comando **pause(n)**, que hará una pausa en la ejecución durante **n** segundos antes de continuar.

El comando **figure** le permite abrir una ventana de figura. La siguiente vez que solicite una gráfica, se desplegará en esta nueva ventana. Por ejemplo,

```
figure(2)
```

abre una ventana llamada figure 2, que luego se convierte en la ventana de uso para subsecuentes graficaciones. En la tabla 5.1 se resumen los comandos que se usan para crear una gráfica simple.

Tabla 5.1 Funciones de graficación básicas

`plot`	Crea una gráfica *x–y*	`plot(x,y)`
`title`	Agrega un título a una gráfica	`title('My Graph')`
`xlabel`	Agrega una etiqueta al eje *x*	`xlabel('Independent Variable')`
`ylabel`	Agrega una etiqueta al eje *y*	`ylabel('Dependent Variable')`

`grid`	Agrega una retícula a la gráfica	`grid`
		`grid on`
		`grid off`

`pause`	Detiene la ejecución del programa, lo que le permite al usuario ver la gráfica	`pause`
`figure`	Determina cuál figura se usará para la gráfica actual	`figure(2)`
`hold`	Congela la gráfica actual, de modo que se puede recubrir una gráfica adicional	`hold on` `hold off`

Gráficas con más de una línea

La creación de una gráfica con más de una línea se puede lograr en muchas formas. Por defecto, la ejecución de un segundo enunciado **plot** borrará la primera gráfica. Sin embargo, puede apilar las gráficas unas encima de otras con el comando **hold on**. Ejecute los siguientes enunciados para crear una gráfica con ambas funciones graficadas en la misma gráfica, como se muestra en la figura 5.4:

```
x = 0:pi/100:2*pi;
y1 = cos(x*4);
plot(x,y1)
y2 = sin(x);
hold on;
plot(x, y2)
```

Los puntos y coma son opcionales tanto en el enunciado **plot** como en el enunciado **hold on**. MATLAB continuará poniendo en capa las gráficas hasta que se ejecute el comando **hold off**:

```
hold off
```

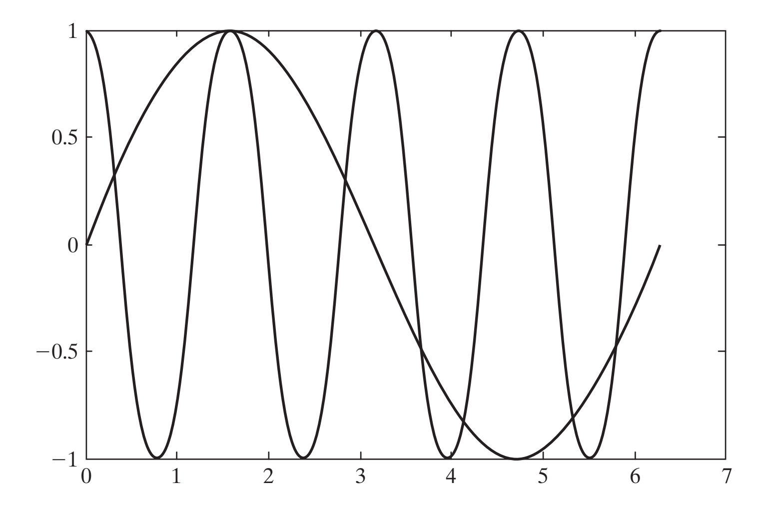


Figura 5.4
Se puede usar el comando **hold on** para poner en capas las gráficas sobre la misma figura.

Idea clave: la gráfica más común usada en ingeniería es la gráfica de dispersión x-y.

Otra forma de crear una gráfica con múltiples líneas es solicitar ambas líneas en un solo comando **plot**. MATLAB interpreta la entrada a **plot** como vectores alternos *x* y *y*, como en

```
plot(X1, Y1, X2,Y2)
```

donde las variables **X1**, **Y1** forman un conjunto ordenado de valores a graficar, y **X2**, **Y2** forman un segundo conjunto ordenado de valores. Con los datos del ejemplo anterior,

```
plot(x,y1, x, y2)
```

produce la misma gráfica que la figura 5.4, con una excepción: las dos líneas tienen colores diferentes. MATLAB usa un color de graficación por defecto (azul) para la primera línea dibujada en un comando **plot**. En el enfoque **hold on**, cada línea se dibuja en un comando plot separado y, por tanto, es el mismo color. Al solicitar dos líneas en un solo comando, como en **plot(x,y1,x,y2)**, la segunda línea por defecto es verde, lo que permite al usuario distinguir entre las dos gráficas.

Si la función **plot** se llama con un solo argumento de matriz, MATLAB dibuja una línea separada para cada columna de la matriz. El eje *x* se etiqueta con el vector índice fila, 1:*k*, donde *k* es el número de filas en la matriz. Esto produce una gráfica igualmente espaciada, a veces llamada gráfica línea. Si **plot** se llama con dos argumentos, uno un vector y el otro una matriz, MATLAB grafica sucesivamente una línea para cada fila en la matriz. Por ejemplo, se pueden combinar **y1** y **y2** en una sola matriz y graficar contra **x**:

```
Y = [y1; y2];
plot(x,Y)
```

Esto crea la misma gráfica que la figura 5.4, con cada línea en un color diferente.

He aquí otro ejemplo más complicado:

```
X = 0:pi/100:2*pi;
Y1 = cos(X)*2;
Y2 = cos(X)*3;
Y3 = cos(X)*4;
Y4 = cos(X)*5;
Z = [Y1; Y2; Y3; Y4];
plot(X, Y1, X, Y2, X, Y3, X, Y4)
```

Este código produce el mismo resultado (figura 5.5) como

```
plot(X, Z)
```

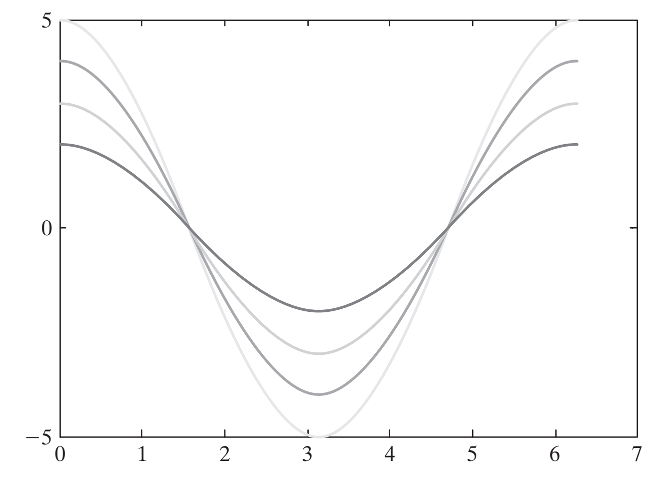


Figura 5.5
Múltiples gráficas en la misma figura.

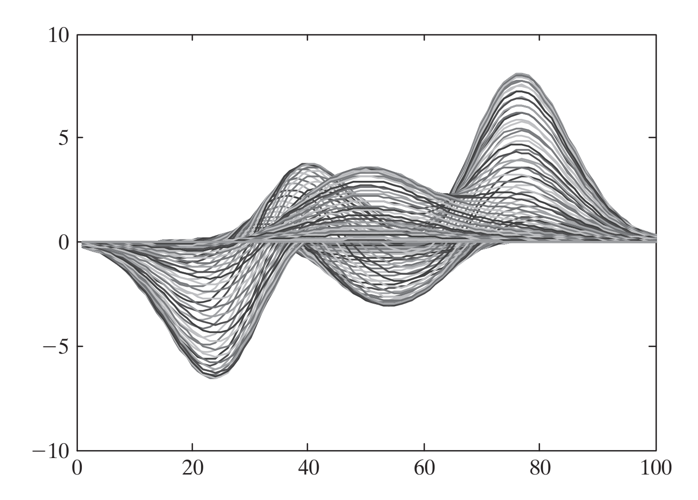


Figura 5.6
La función **peaks**, graficada con un solo argumento en el comando **plot**.

La función **peak** es una función de dos variables que produce datos muestra útiles para demostrar ciertas funciones de graficación. (Los datos se crean al escalar y trasladar distribuciones gaussianas.) Llamar **peaks** con un solo argumento **n** creará una matriz $n \times n$. Se puede usar **peaks** para demostrar la potencia de usar un argumento de matriz en la función **plot**. El comando

```
plot(peaks(100))
```

resulta en la impresionante gráfica de la figura 5.6. La entrada a la función plot creada por peaks es una matriz 100 × 100. Note que el eje *x* va de 1 a 100, los números índice de los datos. Indudablemente usted no lo puede decir, pero hay 100 líneas dibujadas para crear esta gráfica, una por cada columna.

Gráficas de arreglos complejos

Si la entrada al comando **plot** es un arreglo sencillo de números complejos, MATLAB grafica el componente real en el eje *x* y el componente imaginario en el eje *y*. Por ejemplo, si

```
A=[0+0i,1+2i, 2+5i, 3+4i]
```

entonces

```
plot(A)
```

regresa la gráfica que se muestra en la figura 5.7a.

Si se intenta usar dos arreglos de números complejos en la función **plot**, los componentes imaginarios se ignoran. La porción real del primer arreglo se usa para los valores *x* y la porción real del segundo arreglo se usa para los valores *y*. Para ilustrar, cree primero otro arreglo llamado **B** al tomar el seno del arreglo complejo **A**:

```
B=sin(A)
```

regresa

```
B =
   0  3.1658 + 1.9596i  67.4789 -30.8794i  3.8537 -27.0168i
```

y

```
plot(A,B)
```

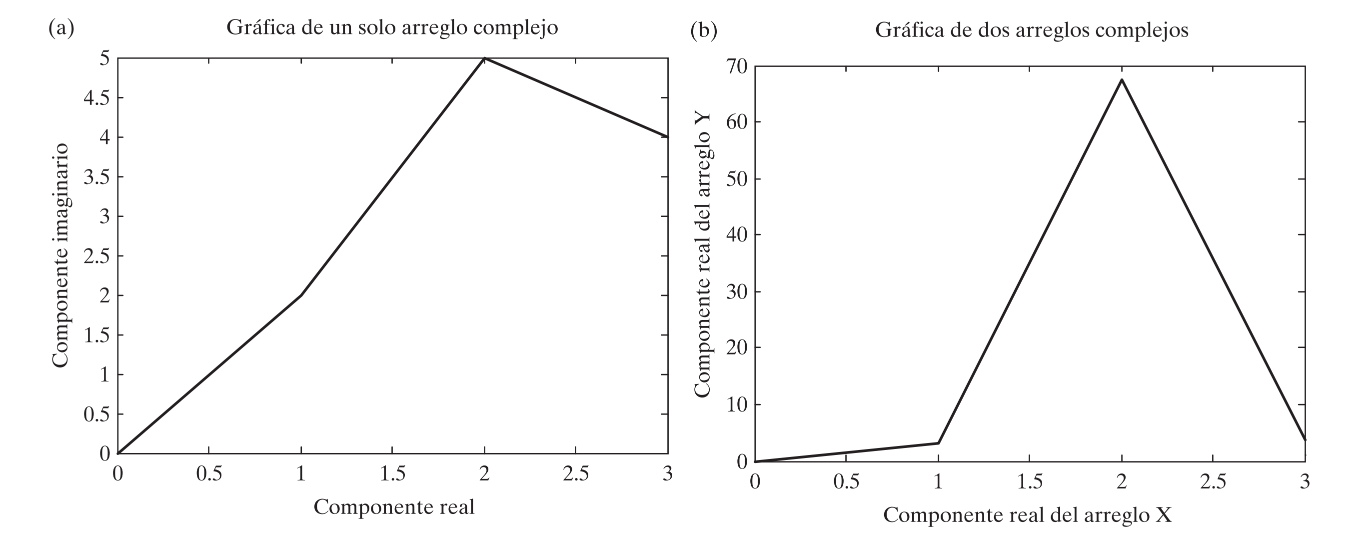


Figura 5.7
(a) Los números complejos se grafican con el componente real sobre el eje *x* y el componente imaginario sobre el eje *y* cuando se usa un solo arreglo como entrada. (b) Cuando en la función **plot** se usan dos arreglos complejos, los componentes imaginarios se ignoran.

da un enunciado de error.

```
Warning: Imaginary parts of complex X and/or Y arguments
ignored.
```

Los datos todavía se grafican, como se muestra en la figura 5.7b.

5.1.2 Línea, color y estilo de marca

Puede cambiar la apariencia de sus gráficas al seleccionar estilo y color de línea definidos por el usuario y al elegir mostrar los puntos de datos en la gráfica con estilos de marca definidos por el usuario. El comando

```
help plot
```

regresa una lista de las opciones disponibles. Puede seleccionar estilos de línea sólido (por defecto), rayado, punteado y raya-punto, y puede escoger mostrar los puntos. Las opciones entre marcas incluyen signos más, estrellas, círculos y marcas x, entre otras. Existen siete diferentes opciones de color. (Véase la tabla 5.2 para una lista completa.)

Los siguientes comandos ilustran el uso de los estilos de línea, color y marca:

```
x = [1:10];
y = [ 58.5, 63.8, 64.2, 67.3, 71.5, 88.3, 90.1, 90.6, 89.5,
    90.4];
plot(x,y,':ok')
```

La gráfica resultante (figura 5.8a) consiste de una línea rayada, junto con puntos de datos marcados con círculos. La línea, los puntos y los círculos se dibujan en negro. Los indicadores se citan dentro de una cadena, denotada con apóstrofes. El orden en el que se ingresan es arbitrario y no afecta la salida.

Tabla 5.2 Opciones de línea, marca y color

Tipo de línea	Indicador	Tipo de punto	Indicador	Color	Indicador
sólida	-	punto	.	azul	b
punteada	:	círculo	o	verde	g
raya-punto	-.	marca x	x	rojo	r
rayada	--	más	+	cian	c
		estrella	*	magenta	m
		cuadrado	s	amarillo	y
		diamante	d	negro	k
		triángulo abajo	v		
		triángulo arriba	^		
		triángulo izquierda	<		
		triángulo derecha	>		
		pentagrama	p		
		hexagrama	h		

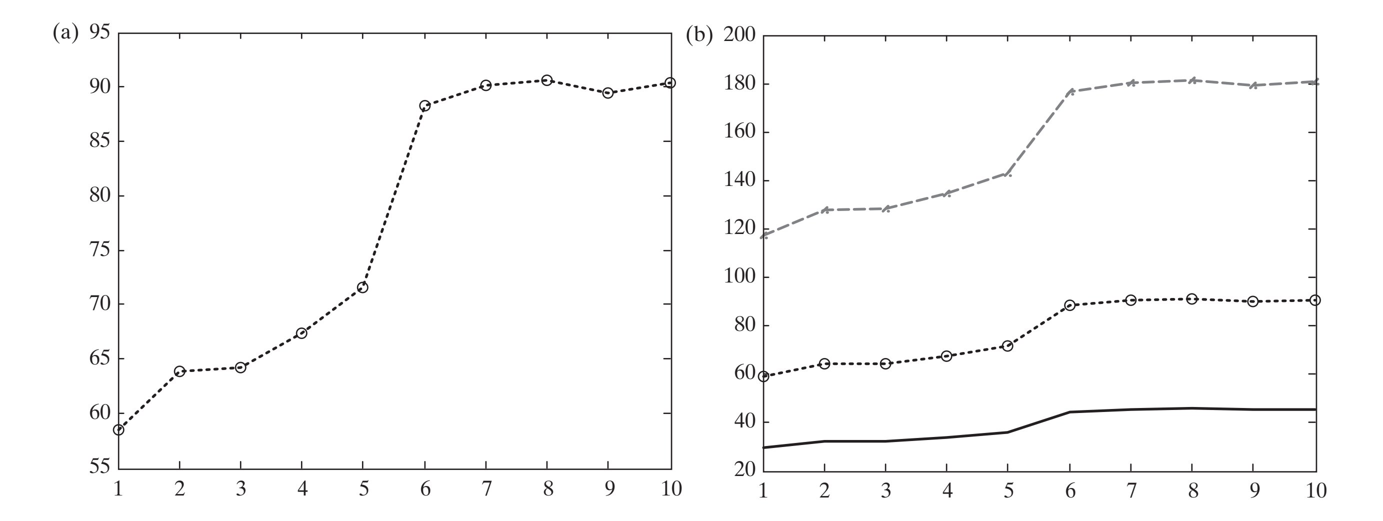


Figura 5.8
(a) Ajustar la línea, marca y color. (b) Múltiples gráficas con variantes de estilos de línea, colores y punto.

Para especificar los estilos de línea, marca y color para múltiples líneas, agregue una cadena que contenga la opciones después de cada par de puntos de datos. Si la cadena no se incluye, se usan los valores por defecto. Por ejemplo,

```
plot(x,y,':ok',x,y*2,'--xr',x,y/2,'-b')
```

resulta en la gráfica que se muestra en la figura 5.8b.

Tabla 5.3 Escalamiento de ejes y anotaciones en gráficas

`axis`	Cuando la función **axis** se usa sin entradas, congela el eje en la configuración actual. Ejecutar la función una segunda vez regresa el control de eje a MATLAB
`axis(v)`	La entrada al comando **axis** debe ser un vector de cuatro elementos que especifique los valores mínimo y máximo para los ejes *x* y *y*, por ejemplo **[xmin,xmax,ymin, ymax]**
`legend('string1', 'string 2', etc)`	Le permite agregar una leyenda a su gráfica. La leyenda muestra un modelo de la línea y menciona la cadena que especificó
`text(x_coordinate,y_coordinate, 'string')`	Le permite agregar un recuadro de texto a la gráfica. El recuadro se coloca en las coordenadas *x* y *y* especificadas y contiene el valor de cadena especificado

El comando **plot** ofrece opciones adicionales para controlar la forma en que aparece la gráfica. Por ejemplo, se puede controlar el ancho de línea. Las gráficas que tienen la intención de verse desde lejos pueden observarse mejor con líneas más gruesas. Use la función **help** para aprender más acerca de cómo controlar la apariencia de la gráfica, o use los controles interactivos que se describen en la sección 5.5.

5.1.3 Escalamiento de ejes y anotaciones en gráficas

MATLAB selecciona automáticamente escalamientos adecuados en los ejes *x* y *y*. A veces es útil para el usuario tener la capacidad de controlar el escalamiento. El control se logra con la función **axis**, que se muestra en la tabla 5.3.

MATLAB ofrece varias funciones adicionales, que también se mencionan en la tabla 5.3, y que le permiten anotar en sus gráficas.

El siguiente código modifica la gráfica de la figura 5.8b con los recuadros **legend** y **text**:

```
legend('línea 1', 'línea 2', 'línea3')
text(1,100,'Etiquete gráficas con el comando text')
```

Se agregó un título, etiquetas *x* y *y*, y se ajusta al eje con los siguientes comandos:

```
xlabel('Mi etiqueta x'), ylabel('Mi etiqueta y')
title('Ejemplo de gráfica para capítulo 5')
axis([0,11,0,200])
```

Los resultados se muestran en la figura 5.9.

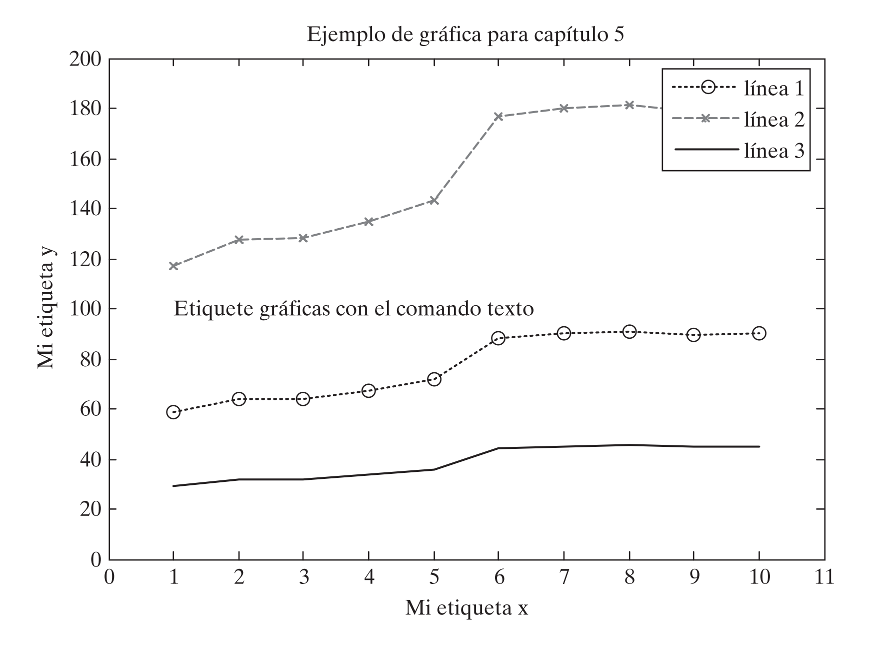


Figura 5.9
Versión final de la gráfica de muestra, anotada con una leyenda, un recuadro de texto, un título, etiquetas *x* y *y*, y un eje modificado.

Sugerencia

Puede usar letras griegas en sus etiquetas al poner una diagonal inversa (\) antes del nombre de la letra. Por ejemplo

```
title('\alpha \beta \gamma')
```

crea el título de la gráfica

$$\alpha\beta\gamma$$

Para crear un superíndice, use llaves. Por tanto,

```
title('x^{2}')
```

produce

$$x^2$$

MATLAB tiene la habilidad de crear expresiones matemáticas más complicadas para usar como títulos, etiquetas de ejes y otras cadenas de texto, al usar el lenguaje formateador de textos TEX. Para aprender más, consulte la característica **help**. (Busque en **TEX** o en **Greek**.)

Ejercicio de práctica 5.1

1. Grafique x contra y para $y = \text{sen}(x)$. Sea x que varía desde 0 hasta 2π en incrementos de 0.1π.
2. Agregue un título y etiquete su gráfica.
3. Grafique x contra y_1 y y_2 para $y_1 = \text{sen}(x)$ y $y_2 = \cos(x)$. Sea x que varía desde 0 hasta 2π en incrementos de 0.1π. Agregue un título y etiquete su gráfica.

4. Vuelva a crear la gráfica de la parte 3, pero haga la línea sen(x) rayada y roja. Haga la línea cos(x) verde y punteada.
5. Agregue una leyenda a la gráfica de la parte 4.
6. Ajuste los ejes de modo que el eje x vaya de -1 a $2\pi + 1$ y el eje y de -1.5 a $+1.5$.
7. Cree un nuevo vector, **a** = cos(**x**). Sea **x** que varía desde 0 hasta 2π en incrementos de 0.1π. Grafique sólo **a** (**plot(a)**) y observe el resultado. Compare este resultado con la gráfica que se produce al graficar **x** contra **a** (**plot(x,a)**).

EJEMPLO 5.1

Uso de la ecuación Clausius-Clapeyron

La ecuación Clausius-Clapeyron se puede usar para encontrar la presión de vapor de saturación del agua en la atmósfera para diferentes temperaturas. La presión de vapor de agua de saturación es útil para los meteorólogos porque se puede usar para calcular humedad relativa, un importante componente de la predicción del clima, cuando se conoce la presión parcial real del agua en el aire.

La siguiente tabla presenta los resultados de calcular la presión de vapor de saturación del agua en la atmósfera a varias temperaturas del aire con el uso de la ecuación Clausius-Clapeyron:

Temperatura del aire	Presión de vapor de saturación
−60.0000	0.0698
−50.0000	0.1252
−40.0000	0.2184
−30.0000	0.3714
−20.0000	0.6163
−10.0000	1.0000
0	1.5888
10.0000	2.4749
20.0000	3.7847
30.0000	5.6880
40.0000	8.4102
50.0000	12.2458
60.0000	17.5747
70.0000	24.8807
80.0000	34.7729
90.0000	48.0098
100.0000	65.5257
110.0000	88.4608
120.0000	118.1931

También presenta estos resultados gráficamente.

La ecuación Clausius-Clapeyron es

$$\ln(P^0/6.11) = \left(\frac{\Delta H_v}{R_{\text{air}}}\right) * \left(\frac{1}{273} - \frac{1}{T}\right)$$

donde

P^0 = presión de vapor de saturación para el agua, en mbar, a temperatura T,
ΔH_v = calor latente de vaporización para el agua, 2.453×10^6 J/kg,
R_v = constante de gas para aire húmedo, 461 J/kg, y
T = temperatura en kelvin

1. Establezca el problema.
   Encontrar la presión de vapor de saturación a temperaturas desde −60 °F hasta 120 °F, con el uso de la ecuación Clausius-Clapeyron.
2. Describa las entradas y salidas.

   Entrada

$$\Delta H_v = 2.453 \times 10^6 \text{ J/kg}$$
$$R_{\text{air}} = 461 \text{ J/kg}$$
$$T = -60\ °\text{F a } 120\ °\text{F}$$

   Dado que el número de valores de temperatura no se especificó, se elegirá recalcular cada 10 °F.

   Salida

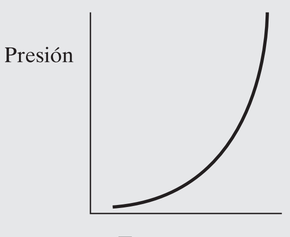
   Tabla de temperatura contra presiones de vapor de saturación
   Gráfica de temperatura contra presiones de vapor de saturación
3. Desarrolle un ejemplo a mano.
   Cambie las temperaturas de Fahrenheit a Kelvin:

$$T_k = \frac{(T_f + 459.6)}{1.8}$$

   Despeje la ecuación Clausius-Clapeyron para la presión de vapor de saturación (P^0):

$$\ln\left(\frac{P^0}{6.11}\right) = \left(\frac{\Delta H_v}{R_{\text{air}}}\right) \times \left(\frac{1}{273} - \frac{1}{T}\right)$$
$$P^0 = 6.11 * \exp\left(\left(\frac{\Delta H_v}{R_{\text{air}}}\right) \times \left(\frac{1}{273} - \frac{1}{T}\right)\right)$$

   Note que la expresión para la presión de vapor de saturación, P^0, es una ecuación exponencial. Por tanto, se esperaría que la gráfica tuviese la siguiente forma:



4. Desarrolle una solución MATLAB.

```
%Ejemplo 5.1
%Con la ecuación Clausius-Clapeyron, encontrar la
%presión de vapor de saturación para agua a diferentes
%temperaturas
%
  TF=[-60:10:120];         %Define matriz temp en F
  TK=(TF + 459.6)/1.8;     %Convierte temp a K
  Delta_H=2.45e6;          %Define calor latente de
                           %vaporización
  R_air = 461;             %Define constante gas ideal
                           %para el aire
```

```
%
%Calcula las presiones de vapor
 Vapor_Pressure = 6.11*exp((Delta_H/R_air)*(1/273 - 1./TK));
%Despliega los resultados en una tabla
 my_results = [TF',Vapor_Pressure']
%
%Crea una gráfica x-y
 plot(TF,Vapor_Pressure)
 title('Comportamiento Clausius-Clapeyron')
 xlabel('Temperatura, F')
 ylabel('Presión de vapor de saturación, mbar')
```

La tabla resultante es

```
my_results =
      -60.0000     0.0698
      -50.0000     0.1252
      -40.0000     0.2184
      -30.0000     0.3714
      -20.0000     0.6163
      -10.0000     1.0000
             0     1.5888
       10.0000     2.4749
       20.0000     3.7847
       30.0000     5.6880
       40.0000     8.4102
       50.0000     12.2458
       60.0000     17.5747
       70.0000     24.8807
       80.0000     34.7729
```

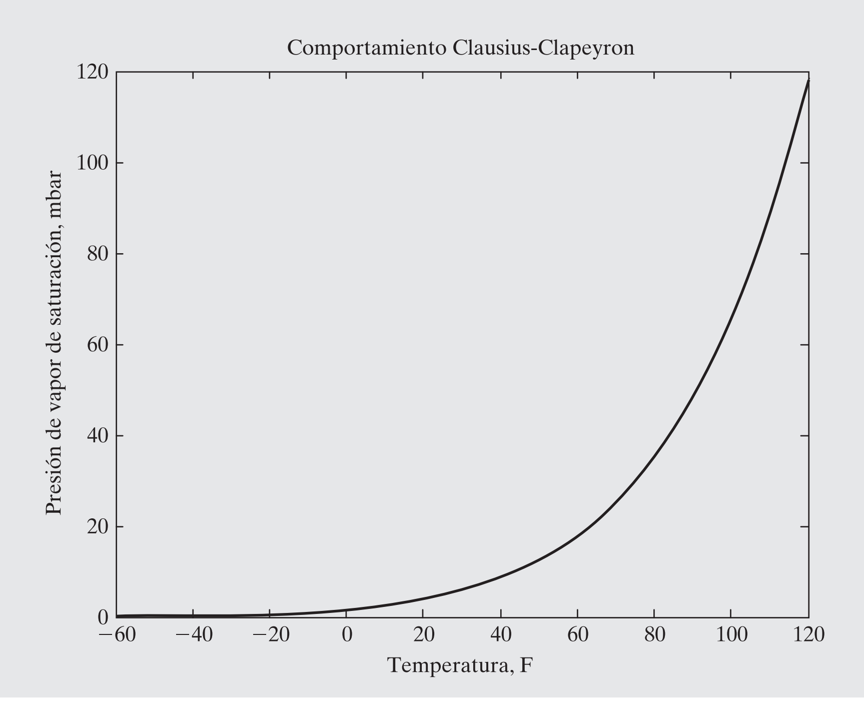


Figura 5.10
Gráfica de la ecuación Clausius-Clapeyron.

```
 90.0000     48.0098
100.0000     65.5257
110.0000     88.4608
120.0000    118.1931
```

Se abre una ventana de figura para desplegar los resultados gráficos, que se muestran en la figura 5.10.

5. Ponga a prueba la solución.
   La gráfica sigue la tendencia esperada. Casi siempre es más fácil determinar si los resultados computacionales tienen sentido si se produce una gráfica. Los datos tabulares son extremadamente difíciles de absorber.

EJEMPLO 5.2

Balística

El rango de un objeto (véase la figura 5.11) que se lanza en un ángulo θ con respecto al eje x y una velocidad inicial v_0 está dado por

$$R(\theta) = \frac{v^2}{g}\text{sen}\,(2\theta) \text{ para } 0 \le \theta \le \frac{\pi}{2} (\text{sin considerar la resistencia del aire})$$

Use $g = 9.9$ m/s^2 y una velocidad inicial de 100 m/s. Demuestre que el rango máximo se obtiene a $\theta = \pi/4$ al calcular y graficar el rango para valores de theta

$$0 \le \theta \le \frac{\pi}{2}$$

en incrementos de 0.05.

Repita sus cálculos con una velocidad inicial de 50 m/s y grafique ambos conjuntos de resultados en una sola gráfica.

1. Establezca el problema.
   Calcular el rango como función del ángulo de lanzamiento.
2. Describa las entradas y salidas.

   Entrada

   g = 9.9 m/s^2
   θ = 0 a $\pi/2$, en incrementos de 0.05
   v_0 = 50 m/s y 100 m/s

   Salida

   Rango R
   Presentar los resultados como gráfica



Figura 5.11
Movimiento balístico.

3. Desarrolle un ejemplo a mano.
   Si el cañón se apunta recto hacia arriba, se sabe que el rango es cero, y si el cañón está horizontal, el rango también es cero. (Véase la figura 5.12.) Esto significa que el rango debe aumentar con el ángulo del cañón hasta algún máximo y luego disminuir. Un cálculo de muestra a 45 grados ($\pi/4$ radianes) muestra que

$$R(\theta) = \frac{v^2}{g}\operatorname{sen}(2\theta)$$

$$R\left(\frac{\pi}{4}\right) = \frac{100^2}{9.9}\operatorname{sen}\left(\frac{2\cdot\pi}{4}\right) = 1010 \text{ metros cuando la velocidad inicial es 100 m/s}$$

4. Desarrolle una solución MATLAB.

```
%Ejemplo 5.2
%El programa calcula el rango de un proyectil balístico
%
% Define las constantes
  g = 9.9;
  v1 = 50;
  v2 = 100;
% Define el vector angle
  angle = 0:0.05:pi/2;
% Calcula el rango
  R1 = v1^2/g*sin(2*angle);
  R2 = v2^2/g*sin(2*angle);
%Grafica los resultados
  plot(angle,R1,angle,R2,':')
  title('Rango del cañón')
  xlabel('Ángulo del cañón')
  ylabel('Rango, metros')
  legend('Velocidad inicial = 50 m/s', 'Velocidad inicial
    = 100 m/s')
```

   Note que el comando **plot** pide a MATLAB imprimir el segundo conjunto de datos como una línea rayada. También se agregaron título, etiquetas y una leyenda. Los resultados se grafican en la figura 5.13.





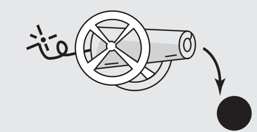


Figura 5.12
El rango es cero si el cañón está perfectamente vertical o perfectamente horizontal.

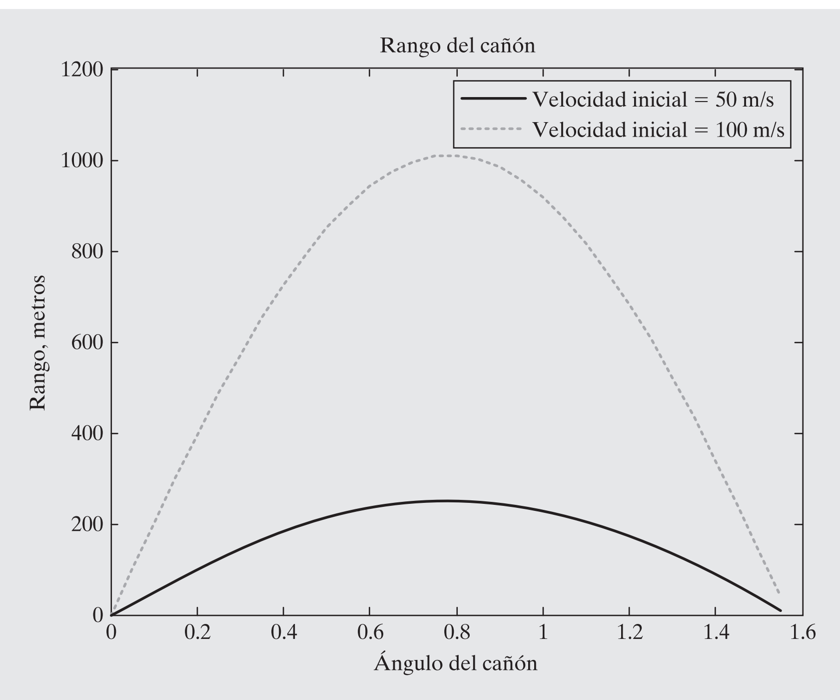


Figura 5.13
Rango predicho de un proyectil.

5. Ponga a prueba la solución.
   Compare los resultados MATLAB con los del ejemplo a mano. Ambas gráficas comienzan y terminan en cero. El rango máximo para una velocidad inicial de 100 m/s es aproximadamente 1000 m, que corresponde bien al valor calculado de 1010 m. Note que ambas soluciones tienen un pico en el mismo ángulo, aproximadamente a 0.8 radianes. El valor numérico para $\pi/4$ es 0.785 radianes, lo que confirma la hipótesis presentada en el enunciado del problema de que el rango máximo se logra al apuntar el cañón en un ángulo de $\pi/4$ radianes (45 grados).

Sugerencia

Para limpiar una figura, use el comando **clf**. Para cerrar una ventana de figura, use el comando **close**.

5.2 SUBGRÁFICAS

El comando **subplot** le permite subdividir la ventana de graficación en una retícula de *m* filas y *n* columnas. La función

```
subplot(m,n,p)
```

separa la figura en una matriz $m \times n$. La variable **p** identifica la porción de la ventana donde se dibujará la siguiente gráfica. Por ejemplo, si se usa el comando

```
subplot(2,2,1)
```

p = 1	p = 2
p = 3	p = 4

Figura 5.14
Las subgráficas se usan para subdividir la ventana de figura en una matriz $m \times n$.

la ventana se divide en dos filas y dos columnas, y la gráfica se dibuja en la ventana superior izquierda (figura 5.14). Las ventanas se numeran de izquierda a derecha, de arriba abajo. De manera similar, los siguientes comandos dividen la ventana de gráficos en una gráfica superior y una gráfica inferior:

```
x=0:pi/20:2*pi;
subplot(2,1,1)
plot(x,sin(x))
subplot(2,1,2)
plot(x,sin(2*x))
```

La primera gráfica se divide en la ventana superior, pues **p** = 1. Luego se usa de nuevo el comando **subplot** para dibujar la siguiente gráfica en la ventana inferior. La figura 5.15 muestra ambas gráficas.

Los títulos se agregan sobre cada subventana conforme las ventanas se dibujan, así como las etiquetas de los ejes *x* y *y*, y cualquier anotación deseada. El uso del comando **subplot** se ilustra en varias de las secciones que siguen.

Ejercicio de práctica 5.2

1. Subdivida una ventana de figura en dos filas y una columna.
2. En la ventana superior, grafique $y = \tan(x)$ para $-1.5 \le x \le 1.5$. Use un incremento de 0.1.
3. Agregue un título y etiquetas de eje a su gráfica.
4. En la ventana inferior, grafique $y = \text{senh}(x)$ para el mismo rango.
5. Agregue un título y etiquetas a su gráfica.
6. Intente de nuevo los ejercicios anteriores, pero divida la ventana de figura verticalmente en lugar de horizontalmente.

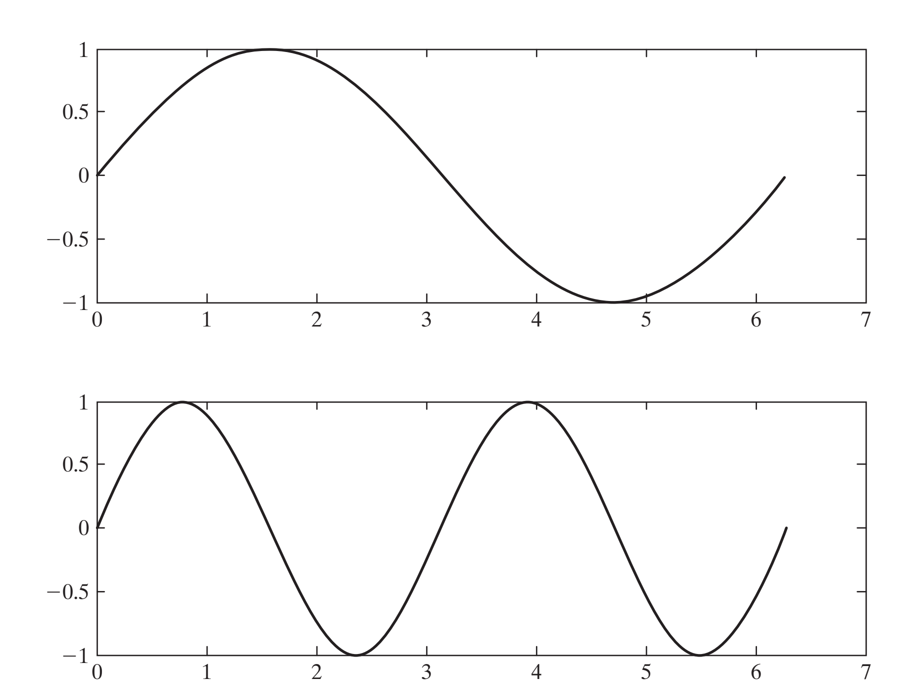


Figura 5.15
El comando **subplot** permite al usuario crear gráficas múltiples en la misma ventana de figura.



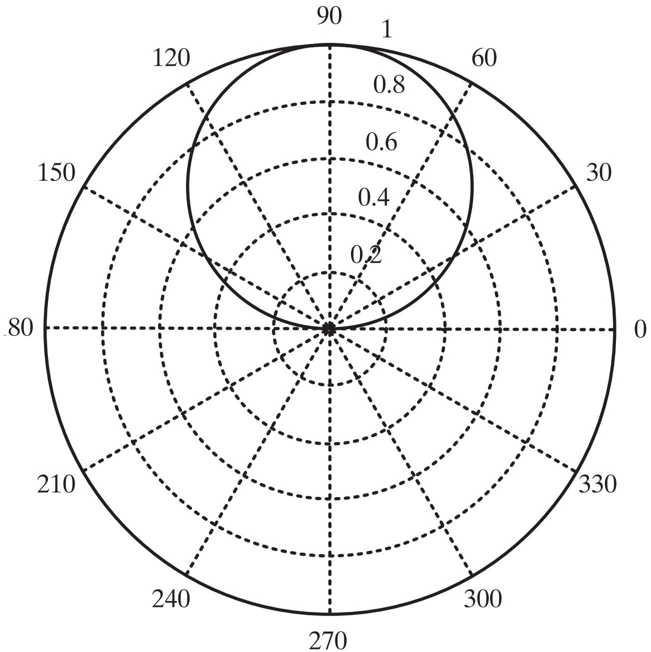


Figura 5.16
Gráfica polar de la función seno.

5.3 OTROS TIPOS DE GRÁFICAS BIDIMENSIONALES

Aunque las gráficas *x-y* simples son el tipo más común de gráfica en ingeniería, existen muchas otras formas de representar datos. Dependiendo de la situación, estas técnicas pueden ser más adecuadas que una gráfica *x-y*.

5.3.1 Gráficas polares

MATLAB proporciona capacidades de graficación con coordenadas polares:

```
polar(theta, r)
```

genera una gráfica polar del ángulo theta (en radianes) y distancia radial *r*.

Por ejemplo, el código

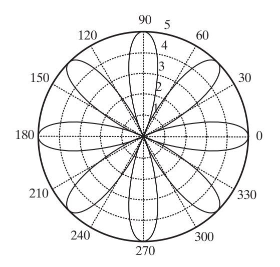
```
x=0:pi/100:pi;
y=sin(x);
polar(x,y)
```

genera la gráfica de la figura 5.16. Se agregó un título en la forma usual:

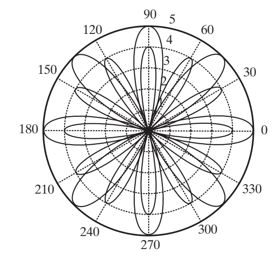
```
title('La función seno graficada en coordenadas polares es
un círculo.')
```

Ejercicio de práctica 5.3

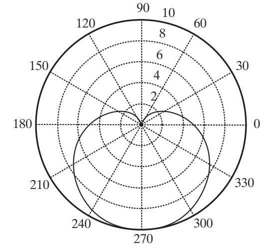
1. Defina un arreglo llamado **theta**, desde 0 hasta 2π, en pasos de 0.01π.
   Defina un arreglo de distancias **r = 5*cos(4*theta)**.
   Elabore una gráfica polar de **theta** contra **r**.



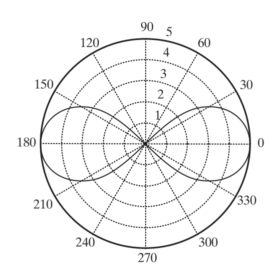
2. Use el comando **hold on** para congelar la gráfica.
   Asigne **r = 4*cos(6*theta)** y grafique.
   Agregue un título.



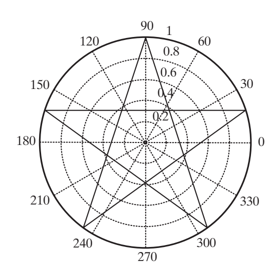
3. Cree una nueva figura.
   Use el arreglo **theta** de los problemas anteriores.
   Asigne **r = 5 – 5*sin(theta)** y cree una nueva gráfica polar.



4. Cree una nueva figura.
   Use el arreglo **theta** de los problemas anteriores.
   Asigne **r = sqrt(5^2*cos(2*theta))** y cree una nueva gráfica polar.



5. Cree una nueva figura.
   Defina un arreglo theta tal que **theta = pi/2:4/5*pi:4.8*pi;**
   Cree un arreglo de seis miembros de unos llamado **r**.
   Cree una nueva gráfica polar de **theta** contra **r**.



5.3.2 Gráficas logarítmicas

Para la mayoría de las gráficas que se generan, los ejes x y y se dividen en intervalos igualmente espaciados; dichas gráficas se llaman gráficas *lineales* o *rectangulares*. Sin embargo, en ocasiones es posible que se quiera usar una escala logarítmica en uno o ambos ejes. Una escala logarítmica (a la base 10) es conveniente cuando una variable varía sobre muchos órdenes de magnitud, porque el amplio rango de valores se puede graficar sin comprimir los valores más pequeños. Las gráficas logarítmicas también son útiles para representar datos que varían exponencialmente.

Idea clave: las gráficas logarítmicas son especialmente útiles si los datos varían exponencialmente.

En la tabla 5.4 se listan los comandos MATLAB para generar gráficas lineales y logarítmicas de los vectores **x** y **y**.

Recuerde que el logaritmo de un número negativo o de cero no existen. Si sus datos incluyen estos valores, MATLAB emitirá un mensaje de advertencia y no se graficarán los puntos en cuestión. Sin embargo, generará una gráfica con base en los puntos restantes.

Cada uno de los comandos para graficación logarítmica se puede ejecutar con un argumento, como se vio en **plot(y)** para una gráfica lineal. En estos casos, las gráficas se generan con los valores de los índices del vector **y** usados como valores **x**.

Como ejemplo, se crearon gráficas de $y = 5x^2$ con los cuatro enfoques de escalamiento, como se muestra en al figura 5.17. Las gráficas lineal (rectangular), semilog en el eje x, semilog en el eje y y log-log se muestran todas en una figura, graficadas con la función **subplot** en el código siguiente:

```
x = 0:0.5:50;
y = 5*x.^2;
subplot(2,2,1)
plot(x,y)
    title('Polinomial - lineal/lineal')
    ylabel('y'), grid
subplot(2,2,2)
semilogx(x,y)
    title('Polinomial - log/lineal')
    ylabel('y'), grid
subplot(2,2,3)
semilogy(x,y)
    title('Polinomial - lineal/log')
    xlabel('x'), ylabel('y'), grid
subplot(2,2,4)
loglog(x,y)
    title('Polinomial - log/log')
    xlabel('x'), ylabel('y'), grid
```

Tabla 5.4 Gráficas rectangular y logarítmica

`plot(x,y)`	Genera una gráfica lineal de los vectores **x** y **y**
`semilogx(x,y)`	Genera una gráfica de los valores de **x** y **y**, con una escala logarítmica para **x** y una escala lineal para **y**
`semilogy(x,y)`	Genera una gráfica de los valores de **x** y **y**, con una escala lineal para **x** y una escala logarítmica para **y**
`loglog(x,y)`	Genera una gráfica de los vectores **x** y **y**, con una escala logarítmica para **x** y **y**

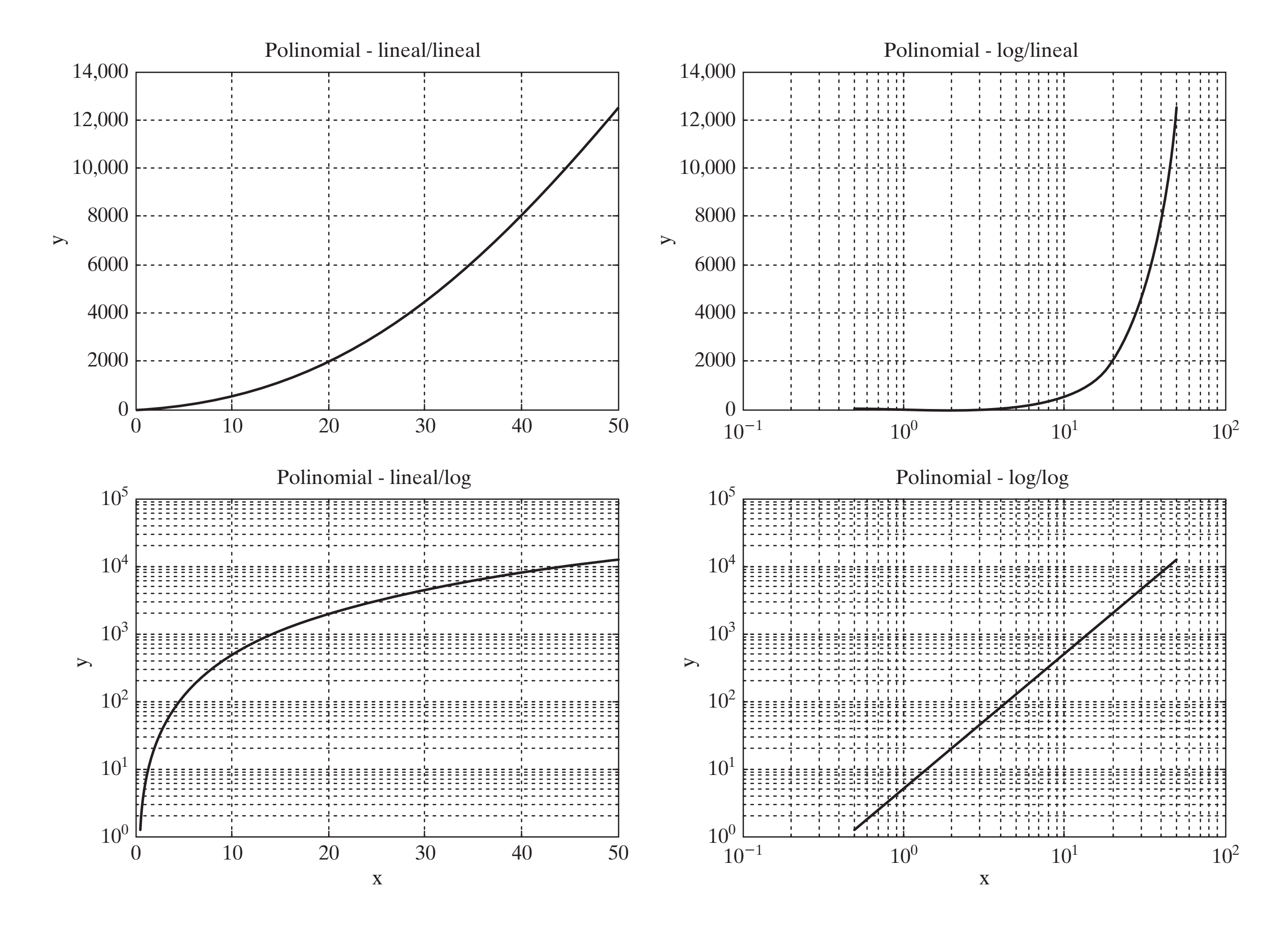


Figura 5.17
Gráficas lineales y logarítmicas.

Idea clave: dado que MATLAB ignora los espacios en blanco, úselos para hacer su código más legible.

Las sangrías tienen la intención de hacer el código más fácil de leer: MATLAB ignora los espacios en blanco. Como cuestión de estilo, note que sólo las dos subgráficas inferiores tienen etiquetas en el eje *x*.

EJEMPLO 5.3

Tasas de difusión

Con frecuencia, los metales se tratan para hacerlos más fuertes y en consecuencia más durables. Un problema con la elaboración de una pieza metálica fuerte es que se vuelve difícil darle la forma deseada. Una estrategia que resuelve este problema es formar un metal suave con la forma que se desea y luego se endurece la superficie. Esto hace que el metal se use bien sin hacerlo quebradizo.

Un proceso de endurecimiento común se llama *carburación*. La parte metálica se expone a carbono, que se difunde en la pieza, haciéndola más dura. Éste es un proceso muy lento

si se realiza a bajas temperaturas, pero se puede acelerar al calentar la pieza. La difusividad es la medida de cuán rápido ocurre la difusión y se puede modelar como

$$D = D_0 \exp\left(\frac{-Q}{RT}\right)$$

donde

D = difusividad, cm^2/s,
D_0 = coeficiente de difusión, cm^2/s,
Q = energía de activación, J/mol, 8.314 J/mol K,
R = constante de gas ideal, J/mol K, y
T = temperatura, K.

Conforme el hierro se calienta, cambia su estructura y sus características de difusión. Los valores de D_0 y Q se muestran en la siguiente tabla para difusión de carbono a través de cada una de las estructuras de hierro:

Tipo de metal	$\mathbf{D_0}$ **(cm^2/s)**	$\mathbf{Q}$ **(J/mol K)**
Fe alfa (BCC)	.0062	80,000
Fe gamma (FCC)	0.23	148,000

Con los datos proporcionados, cree una gráfica de difusividad contra temperatura inversa ($1/T$). Intente las gráficas rectangular, semilog y log-log para ver cuál puede representar mejor los resultados. Haga que la temperatura varíe de temperatura ambiente (25 °C) a 1200 °C.

1. Establezca el problema.
   Calcular la difusividad del carbono en hierro.
2. Describa las entradas y salidas.

   Entrada

   Para C en hierro alfa, D_0 = 0.0062 cm^2/s y Q = 80,000 J/mol K
   Para C en hierro gamma, D_0 = 0.23 cm^2/s y Q = 148,000 J/mol K
   R = 8.314 J/mol K
   T varía de 25 °C a 1200 °C

   Salida

   Calcular la difusividad y graficarla
3. Desarrolle un ejemplo a mano.
   La difusividad está dada por

$$D = D_0 \exp\left(\frac{-Q}{RT}\right)$$

   A temperatura ambiente, la difusividad para el carbono en hierro alfa es

$$D = .0062 \exp\left(\frac{-80{,}000}{8.314 \cdot (25 + 273)}\right)$$
$$D = 5.9 \times 10^{-17}$$

   (Note que la temperatura tuvo que cambiarse de Celsius a Kelvin.)

4. Desarrolle una solución MATLAB.

```
% Ejemplo 5.3
% Calcula la difusividad de carbono en hierro
  clear, clc
% Define las constantes
  D0alpha = .0062;
  D0gamma = 0.23;
  Qalpha = 80000;
  Qgamma = 148000;
  R = 8.314;
  T = 25:5:1200;
% Cambia T de C a K
  T = T+273;
% Calcula la difusividad
  Dalpha = D0alpha*exp(-Qalpha./(R*T));
  Dgamma = D0gamma*exp(-Qgamma./(R*T));
% Grafica los resultados
  subplot(2,2,1)
  plot(1./T,Dalpha, 1./T,Dgamma)
  title('Difusividad de C en Fe')
  xlabel('Temperatura inversa, K^-1'),ylabel('Difusividad,
    cm^2/s')
  grid on

  subplot(2,2,2)
  semilogx(1./T,Dalpha, 1./T,Dgamma)
  title('Difusividad de C en Fe')
  xlabel('Temperatura inversa, K^-1'),ylabel('Difusividad,
   cm^2/s')
  grid on

  subplot(2,2,3)
  semilogy(1./T,Dalpha, 1./T,Dgamma)
  title('Difusividad de C en Fe')
  xlabel('Temperatura inversa, K^-1'),ylabel('Difusividad,
   cm^2/s')
  grid on

  subplot(2,2,4)
  loglog(1./T,Dalpha, 1./T,Dgamma)
  title('Difusividad de C en Fe')
  xlabel('Temperatura inversa, K^-1'),ylabel('Difusividad,
   cm^2/s')
  grid on
```

En la figura 5.18 se usaron subgráficas, de modo que todas las variaciones de la gráfica están en la misma figura. Note que las etiquetas *x* se agregaron sólo en la parte inferior de dos gráficas, para reducir el amontonamiento, y que sólo a la primera gráfica se agregó una leyenda. La gráfica **semilogy** resultó en líneas rectas y permite al usuario leer los valores de la gráfica fácilmente sobre un amplio rango de temperaturas y difusividades. Éste es el esquema de graficación que usualmente se usa para presentar los valores de difusividad en libros de texto y manuales.

5. Ponga a prueba la solución.
   Compare los resultados MATLAB con los del ejemplo a mano.

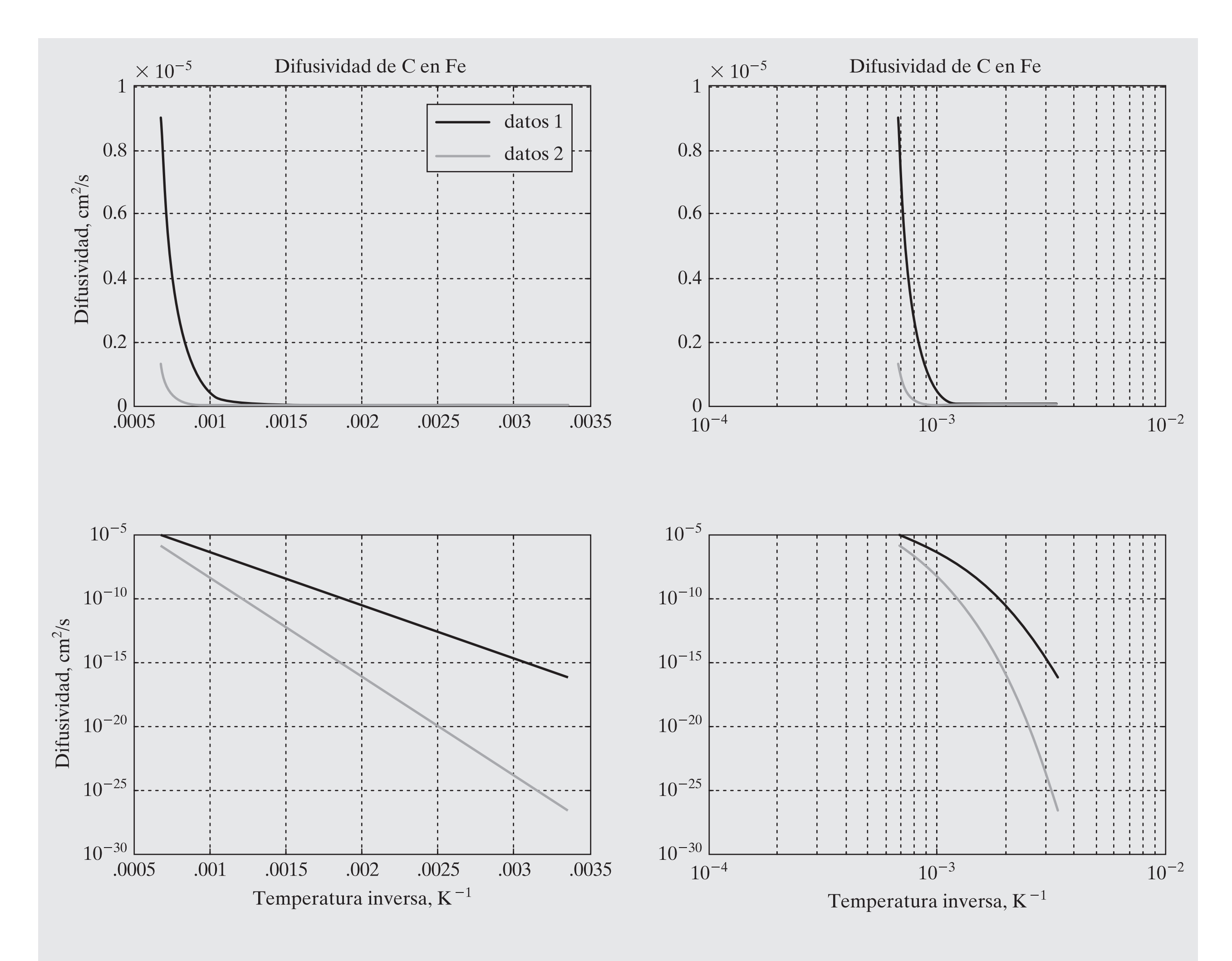


Figura 5.18
Datos de difusividad graficados en diferentes escalas.

Se calculó que la difusividad era

$$5.9 \times 10^{-17} \text{ cm}^2\text{/s en } 25\ °\text{C}$$

para carbono en hierro alfa. Para verificar su respuesta, necesitará cambiar 25 °C a kelvins y sacar el inverso:

$$\frac{1}{(25 + 273)} = 3.36 \times 10^{-3}$$

A partir de la gráfica **semilogy** (esquina inferior izquierda), se puede ver que la difusividad para el hierro alfa es aproximadamente 10^{-17}.

Ejercicio de práctica 5.4

Cree arreglos **x** y **y** adecuados para usar en gráfica cada una de las expresiones que siguen. Use el comando **subplot** para dividir sus figuras en cuatro secciones y cree cada una de estas cuatro gráficas para cada expresión:

- rectangular
- semilogx
- semilogy
- loglog

1. $y = 5x + 3$
2. $y = 3x^2$
3. $y = 12e^{(x+2)}$
4. $y = 1/x$

Usualmente, los datos físicos se grafican de modo que caen en una línea recta. ¿Cuál de los tipos de gráficas precedentes resultan en una línea recta para cada problema?

5.3.3 Gráficas de barras y de pastel

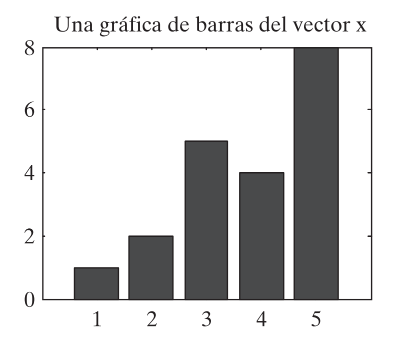
Las gráficas de barra, histograma y de pastel son formas populares para reportar datos. En la tabla 5.5 se mencionan algunas de las funciones MATLAB usadas comúnmente para crear gráficas de barra y de pastel.

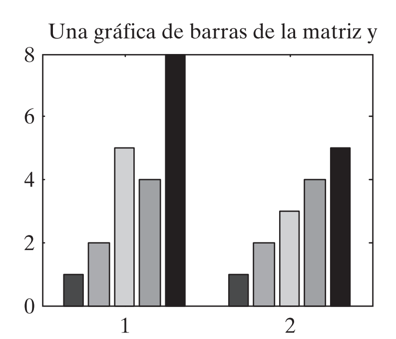
En la figura 5.19 se muestran ejemplos de algunas de estas gráficas. Las gráficas usan la función **subplot** para permitir cuatro gráficas en la misma ventana de figura:

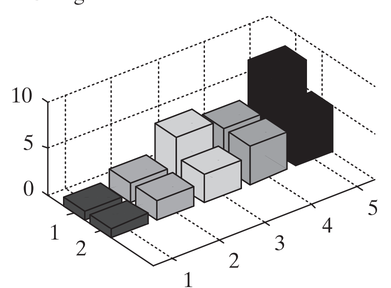
```
clear, clc
x=[1,2,5,4,8];
y=[x;1:5];
```

Tabla 5.5 Gráficas de barra y de pastel

bar(x)	Cuando **x** es un vector, **bar** genera una gráfica de barras vertical. Cuando **x** es una matriz bidimensional, **bar** agrupa los datos por fila
barh(x)	Cuando **x** es un vector, **barh** genera una gráfica de barras horizontal Cuando **x** es una matriz bidimensional, **barh** agrupa los datos por fila
bar3(x)	Genera una gráfica de barras tridimensional
bar3h(x)	Genera una gráfica de barras horizontal tridimensional
pie(x)	Genera una gráfica de pastel. Cada elemento en la matriz se representa como una rebanada de pastel
pie3(x)	General una gráfica de pastel tridimensional. Cada elemento en la matriz se representa como una rebanada de pastel
hist(x)	Genera un histograma







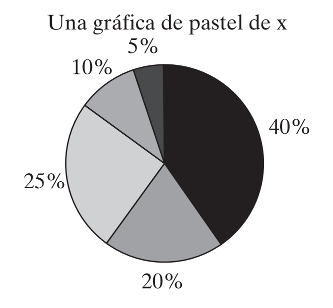


Figura 5.19
Ejemplo de gráficas bidimensionales que usan la función **subplot** para dividir la ventana en cuadrantes.

```
subplot(2,2,1)
  bar(x),title('Una gráfica de barras del vector x')
subplot(2,2,2)
  bar(y),title('Una gráfica de barras de la matriz y')
subplot(2,2,3)
  bar3(y),title('Una gráfica de barras tridimensional')
subplot(2,2,4)
  pie(x),title('Una gráfica de pastel de x')
```

5.3.4 Histogramas

Idea clave: los histogramas son útiles en análisis estadístico.

Un histograma es un tipo especial de gráfica particularmente útil para el análisis estadístico de datos. Es una gráfica que muestra la distribución de un conjunto de valores. En MATLAB, el histograma calcula el número de valores que caen en 10 depósitos (categorías) que están igualmente espaciadas entre los valores mínimo y máximo. Por ejemplo, si se define una matriz **x** como el conjunto de calificaciones del Final de Introducción a la ingeniería, las calificaciones se podrían representar en un histograma, que se muestra en la figura 5.20 y se genera con el siguiente código:

```
x=[100,95,74,87,22,78,34,35,93,88,86,42,55,48];
hist(x)
```

El número por defecto de depósitos (bins) es 10, pero si se tiene un gran conjunto de datos, se pueden dividir los datos en más depósitos. Por ejemplo, para crear un histograma con 25 depósitos, el comando sería

```
hist(x, 25)
```

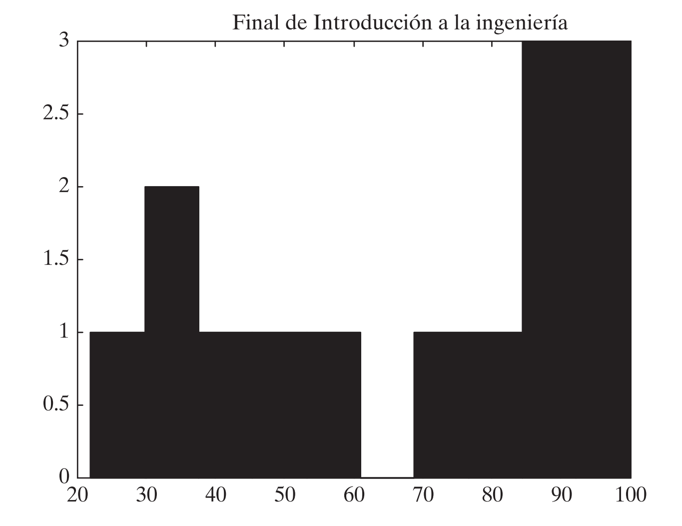


Figura 5.20
Histograma de datos de calificación.

Si establece la función **hist** igual a una variable, como en

```
A = hist(x)
```

los datos que se usan en la gráfica se almacenan en **A**:

```
A =
   1   2   1   1   1   0   1   1   3   3
```

EJEMPLO 5.4

Distribuciones de peso

El varón estadounidense promedio de 18 años de edad pesa 152 libras. Se pesa a un grupo de 100 jóvenes y los datos se almacenan en un archivo llamado **weight.dat**. Cree una gráfica para representar los datos.

1. Establezca el problema.
   Usar el archivo de datos para crear una gráfica lineal y un histograma. ¿Cuál es una mejor representación de los datos?
2. Describa las entradas y salidas.

   Entrada **weight.dat**, un archivo de datos ASCII que contiene datos de peso

   Salida Una gráfica lineal de los datos
   Un histograma de los datos
3. Desarrolle un ejemplo a mano.
   Dado que ésta es una muestra de pesos reales, se esperaría que los datos se aproximaran a una distribución aleatoria normal (una distribución gaussiana). El histograma debería tener forma de campana.
4. Desarrolle una solución MATLAB.
   El siguiente código genera las gráficas que se muestran en la figura 5.21:

```
% Ejemplo 5.4
% Uso de datos de peso
%
load weight.dat
```

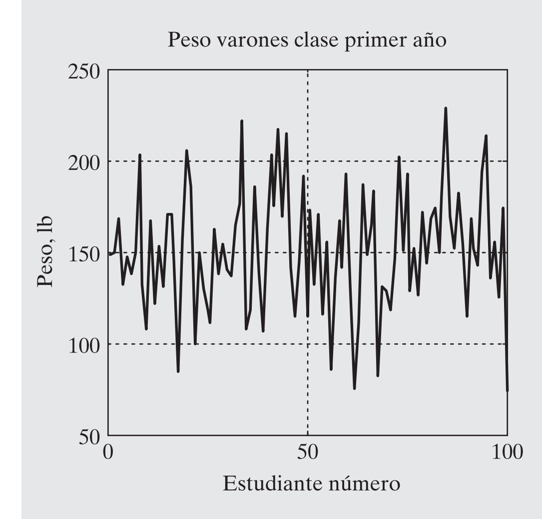
```
% Crea la gráfica lineal de datos de peso
subplot(1,2,1)
plot(weight)
title('Peso varones clase primer año')
xlabel('Estudiante número')
ylabel('Peso, lb')
grid on
% Crea el histograma de los datos
subplot(1,2,2)
hist(weight)
xlabel('Peso, lb')
ylabel('Número de estudiantes')
title('Peso varones clase primer año')
```

5. Ponga a prueba la solución.
   Las gráficas satisfacen las expectativas. El peso parece promediar aproximadamente 150 lb y varía en lo que parece ser una distribución normal. Se puede usar MATLAB para encontrar el promedio y la desviación estándar de los datos, así como los pesos máximo y mínimo en el conjunto de datos. El código MATLAB

```
average_weight = mean(weight)
standard_deviation = std(weight)
maximum_weight = max(weight)
minimum_weight = min(weight)
```

regresa

```
average_weight =
  151.1500
standard_deviation =
  32.9411
maximum_weight =
  228
minimum_weight =
  74
```



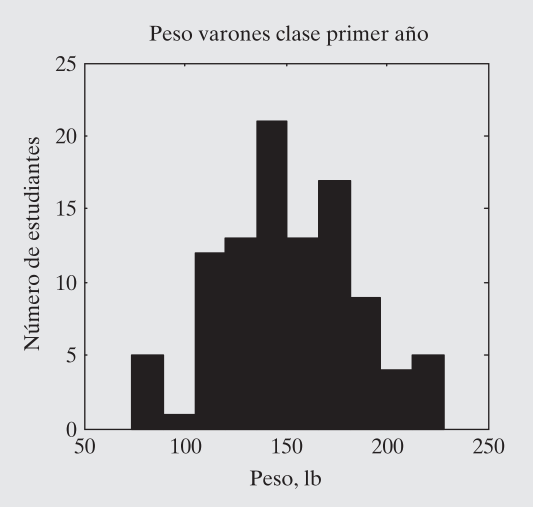


Figura 5.21
Los histogramas y las gráficas de línea son dos formas diferentes de visualizar información numérica.

5.3.5 Gráficas *x-y* con dos ejes *y*

A veces es útil sobreponer dos gráficas *x-y* en la misma figura. Sin embargo, si los órdenes de magnitud de los valores *y* son muy diferentes, puede ser difícil ver cómo se comportan los datos. Considere, por ejemplo, una gráfica de sen(x) y e^x dibujadas en la misma figura. Los resultados, obtenidos con el siguiente código, se muestran en la figura 5.22:

```
x=0:pi/20:2*pi;
y1=sin(x);
y2=exp(x);
subplot(2,1,1)
plot(x,y1,x,y2)
```

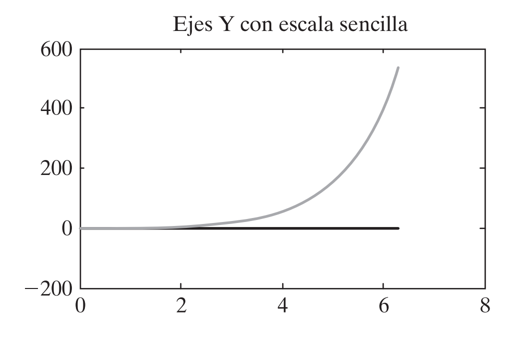
La gráfica de sen(x) parece que corre en línea recta a lo largo de $x = 0$, debido a la escala. La función **plotyy** le permite crear una gráfica con dos ejes *y*, el de la izquierda para el primer conjunto de pares ordenados y el de la derecha para el segundo conjunto de pares ordenados:

```
subplot(2,1,2)
plotyy(x,y1,x,y2)
```

Los títulos y etiquetas se agregaron en la forma usual. El eje *y* no se etiquetó, porque los resultados son adimensionales.

La función **plotyy** puede crear algunos tipos diferentes de gráficas al agregar una cadena con el nombre del tipo de gráfica después del segundo conjunto de pares ordenados. En la figura 5.23, las gráficas se crearon con el siguiente código y tienen un eje con escala logarítmica:

```
subplot(2,1,1)
plotyy(x,y1,x,y2, 'semilogy')
subplot(2,1,2)
plotyy(x,y1,x,y2,'semilogx')
```



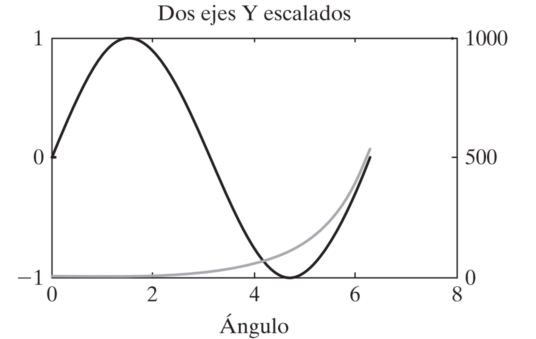
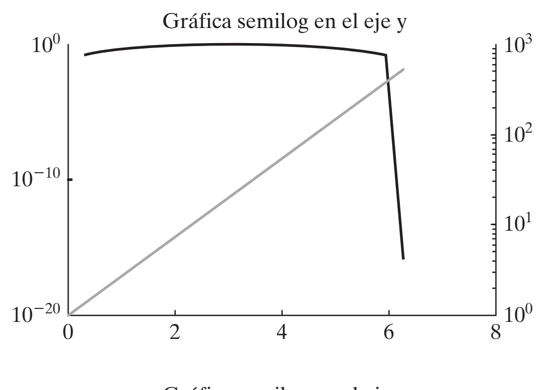


Figura 5.22
MATLAB permite que el eje *y* se escale de manera diferente a la izquierda y derecha de la figura.



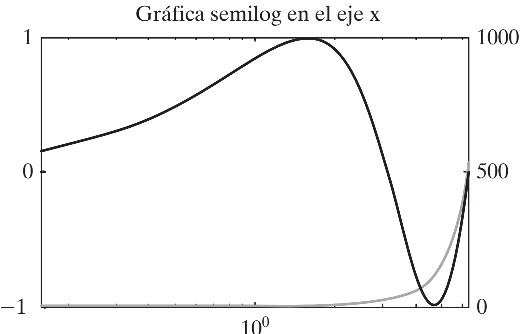


Figura 5.23
La función **plotyy** puede generar varios tipos de gráficas, incluidas semilogx, semilogy y loglog.

EJEMPLO 5.5

Propiedades periódicas de los elementos

Las propiedades de los elementos en la misma fila o columna en la tabla periódica, por lo general, muestran una tendencia reconocible conforme uno se mueve a través de una fila o por una columna. Por ejemplo, el punto de fusión, por lo general, baja conforme se desciende por una columna, porque los átomos están más separados y, por tanto, los enlaces entre los átomos son más débiles. De igual modo, el radio de los átomos aumenta conforme desciende por una columna, porque existen más electrones en cada átomo y en correspondencia orbitales más grandes. Es instructivo graficar estas tendencias contra peso atómico en la misma gráfica.

1. Establezca el problema.
   Graficar el punto de fusión y el radio atómico de los elementos del Grupo I contra el peso atómico y comentar acerca de las tendencias que se observen.
2. Describa las entradas y salidas.

   Entrada Los pesos atómicos, puntos de fusión y radios atómicos de los elementos del Grupo I, que se mencionan en la tabla 5.6

   Salida Graficar punto de fusión y radio atómico en la misma gráfica

3. Desarrolle un ejemplo a mano.
   Se esperaría que la gráfica se parezca al bosquejo que se muestra en la figura 5.24.

Tabla 5.6 Elementos del Grupo I y propiedades físicas seleccionadas

Elemento	Número atómico	Punto de fusión, °C	Radio atómico, pm
Litio	3	181	0.1520
Sodio	11	98	0.1860
Potasio	19	63	0.2270
Rubidio	37	34	0.2480
Cesio	55	28.4	0.2650

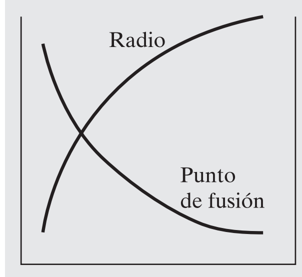


Figura 5.24
Bosquejo del comportamiento predicho de los datos.

4. Desarrolle una solución MATLAB.
   El siguiente código produce la gráfica que se muestra en la figura 5.25:

```
% Ejemplo 5.5
clear, clc
% Define las variables
atomic_number = [ 3, 11, 19, 37, 55];
melting_point = [181, 98, 63, 34, 28.4];
atomic_radius = [0.152, 0.186, 0.227, 0.2480, 0.2650];
% Crea la gráfica con dos líneas en la misma escala
subplot(1,2,1)
plot(atomic_number,melting_point,atomic_number,atomic_radius)
title('Periodic Properties')
xlabel('Atomic Number')
ylabel('Properties')
% Crea la segunda gráfica con dos diferentes escalas y
subplot(1,2,2)
plotyy(atomic_number,melting_point,atomic_number,
  atomic_radius)
title('Periodic Properties')
xlabel('Atomic Number')
ylabel('Melting Point, C')
```

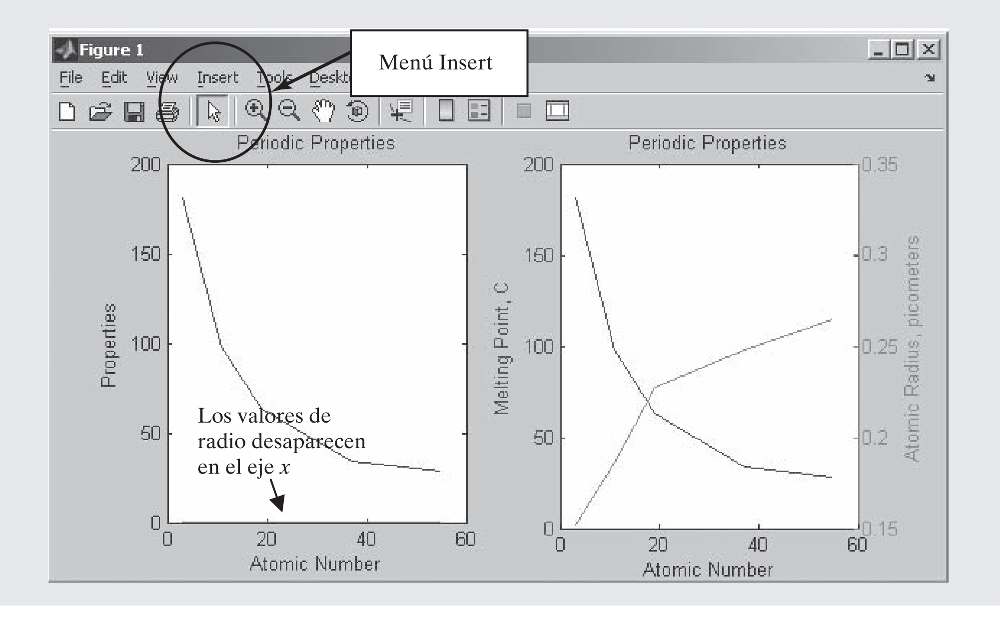


Figura 5.25
El uso de dos ejes *y* le permite graficar datos con unidades diferentes en la misma gráfica.

En la segunda gráfica, que tiene dos diferentes escalas *y*, se usó la función **plotyy** en lugar de la función **plot**. Esto forzó la adición de una segunda escala, en el lado derecho de la gráfica. Fue necesario porque el radio atómico y el punto de fusión tienen diferentes unidades y los valores para cada uno tienen diferentes magnitudes. Note que, en la primera gráfica, casi es imposible ver la línea de radio atómico; está arriba del eje *x* porque los números son muy pequeños. Es posible, pero difícil, agregar la etiqueta del eje *y* derecho desde la línea de comando. En vez de ello, se usó la opción **Insert** de la barra de menú. Sólo recuerde: si vuelve a correr su programa, perderá la etiqueta del lado derecho.

5. Ponga a prueba la solución.
   Compare los resultados MATLAB con los del ejemplo a mano. La tendencia se ajusta a la predicción. Claramente, la gráfica con dos ejes *y* es la representación superior, porque se pueden ver las tendencias de la propiedad.

5.3.6 Gráficas de función

La función **fplot** le permite graficar una función sin definir arreglos de valores *x* y *y* correspondientes. Por ejemplo,

```
fplot('sin(x)',[-2*pi,2*pi])
```

crea una gráfica (figura 5.26) de *x* contra sen(*x*) para valores *x* desde -2π hasta 2π. MATLAB calcula automáticamente el espaciamiento de los valores *x* para crear una curva suave. Note que el primer argumento en la función **fplot** es una cadena que contiene la función y el segundo argumento es un arreglo.

Ejercicio de práctica 5.5

Cree una gráfica de las funciones que siguen. Necesitará seleccionar un rango apropiado para cada gráfica. No olvide poner título y etiquetas a sus gráficas.

1. $f(t) = 5t^2$
2. $f(t) = 5\,\text{sen}^2(t) + t\cos^2(t)$
3. $f(t) = te^t$
4. $f(t) = \ln(t) + \text{sen}(t)$

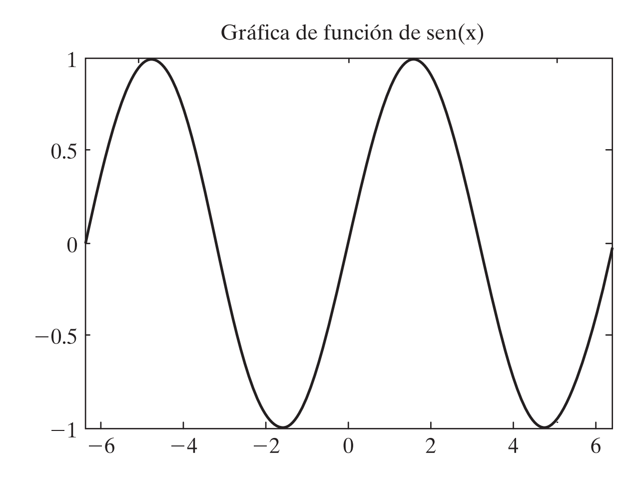


Figura 5.26
Las gráficas de función no requieren que el usuario defina arreglos de pares ordenados.

Sugerencia

La sintaxis correcta para la expresión matemática $\text{sen}^2(t)$ es **sin(t).^2**.

5.4 GRÁFICAS TRIDIMENSIONALES

MATLAB ofrece una variedad de comandos para gráficas tridimensionales, muchas de las cuales se mencionan en la tabla 5.7.

5.4.1 Gráfica lineal tridimensional

La función **plot3** es similar a la función **plot**, excepto que acepta datos en tres dimensiones. Sin embargo, en lugar de sólo proporcionar vectores **x** y **y**, el usuario también debe proporcionar un vector **z**. Entonces estas tripletas ordenadas se grafican en un espacio de tres dimensiones y se conectan con líneas rectas. Por ejemplo

```
clear, clc
x = linspace(0,10*pi,1000);
y = cos(x);
z = sin(x);
plot3(x,y,z)
grid
xlabel('ángulo'), ylabel('cos(x)'), zlabel('sen(x)'),
  title('Un resorte')
```

El título, las etiquetas y la retícula se agregaron a la gráfica de la figura 5.27 en la forma usual, con la adición de **zlabel** para el eje z.

El sistema coordenado usado con **plot3** se orienta mediante el sistema coordenado de la mano derecha familiar a los ingenieros.

Tabla 5.7 Gráficas tridimensionales

`plot3(x,y,z)`	Crea una gráfica lineal tridimensional
`comet3(x,y,z)`	Genera una versión animada de **plot3**
`mesh(z) o mesh(x,y,z)`	Crea una gráfica de superficie de malla
`surf(z) o surf(x,y,z)`	Crea una gráfica de superficie; similar a la función mesh
`shading interp`	Interpola entre los colores usados para ilustrar gráficas de superficie
`shading flat`	Colorea cada sección de retícula con un color sólido
`colormap(map_name)`	Permite al usuario seleccionar el patrón de color a usar en las gráficas de superficie
`contour(z) o contour(x,y,z)`	Genera una gráfica de contorno
`surfc(z) o surfc(x,y,z)`	Crea una gráfica de superficie combinada con una gráfica de contorno
`pcolor(z) o pcolor(x,y,z)`	Crea una gráfica en pseudocolor

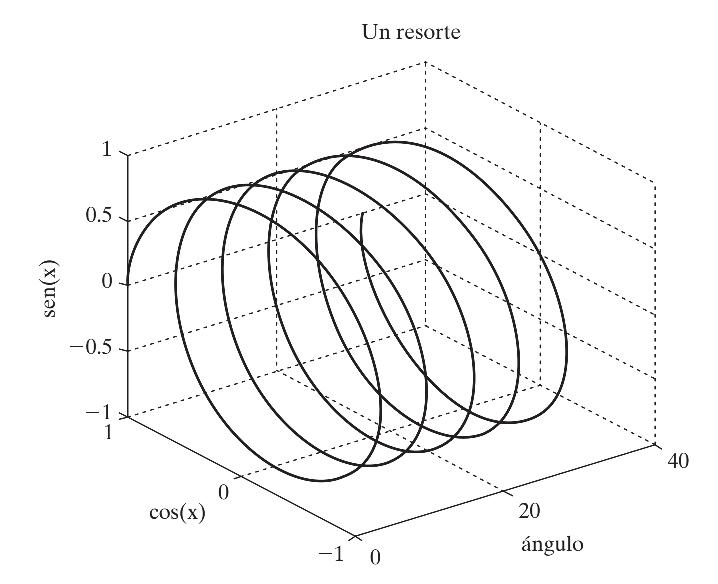


Figura 5.27
Gráfica tridimensional de un resorte.

Idea clave: los ejes que se usan para la graficación tridimensional corresponden a la regla de la mano derecha.

> **Sugerencia**
>
> Sólo por diversión, vuelva a crear la gráfica que se muestra en la figura 5.27, pero esta vez con la función **comet3**:
>
> ```
> comet3(x,y,z)
> ```
>
> Esta función de graficación "dibuja" la gráfica en una secuencia animada. Si su animación corre demasiado rápido, agregue más puntos de datos. Para gráficas lineales bidimensionales, use la función **comet**.

5.4.2 Gráficas de superficie

Las gráficas de superficie le permiten representar datos como una superficie. Se experimentará con dos tipos de gráficas de superficie: gráficas **mesh** y gráficas **surf**.

Gráficas mesh

Existen muchas formas de usar las gráficas **mesh** (malla). Se pueden usar para dar buen efecto a una matriz bidimensional sencilla $m \times n$. En esta aplicación, el valor en la matriz representa el valor **z** en la gráfica. Los valores **x** y **y** se basan en las dimensiones de la matriz. Tome, por ejemplo, la siguiente matriz muy simple:

```
z = [1, 2, 3, 4,  5,  6,  7,  8,  9, 10;
     2, 4, 6, 8, 10, 12, 14, 16, 18, 20;
     3, 4, 5, 6,  7,  8,  9, 10, 11, 12];
```

El código

```
mesh(z)
xlabel('eje x')
ylabel('eje y')
zlabel('eje z')
```

genera la gráfica de la figura 5.28.

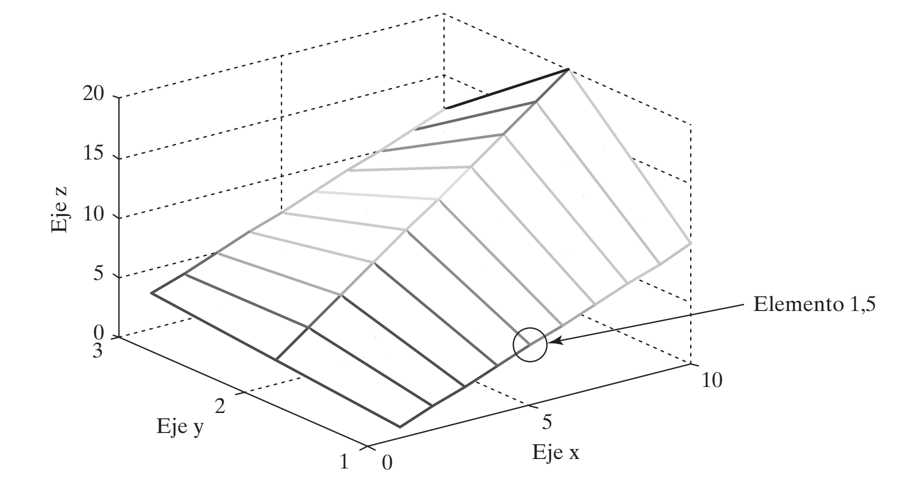


Figura 5.28
Malla simple creada con una matriz bidimensional simple.

La gráfica es una "malla" creada al conectar los puntos definidos en **z** en una retícula rectilínea. Note que el eje *x* va de 0 a 10 y *y* va de 0 a 3. Los números índice de matriz se usaron para los valores de eje. Por ejemplo, note que $z_{1,5}$, el valor de *z* en la fila 1 columna 5, es igual a 5. Este elemento está en un círculo en la figura 5.28.

La función **mesh** también se puede usar con tres argumentos: **mesh(x,y,z)**. En este caso, **x** es una lista de coordenadas *x*, **y** es una lista de coordenadas *y*, y **z** es una lista de coordenadas *z*.

```
x = linspace(1,50,10)
y = linspace(500,1000,3)
z = [1, 2, 3, 4,  5,   6,  7,  8,  9, 10;
     2, 4, 6, 8, 10,  12, 14, 16, 18, 20;
     3, 4, 5, 6,  7,   8,  9, 10, 11, 12]
```

El vector **x** debe tener el mismo número de elementos que el número de columnas en el vector **z**; el vector **y** debe tener el mismo número de elementos que el número de filas en el vector **z**. El comando

```
mesh(x,y,z)
```

crea la gráfica de la figura 5.29a. Note que el eje *x* varía de 0 a 60, con datos graficados de 1 a 50. Compare este escalamiento con el de la figura 5.28, que los números índice de la matriz **z** para los ejes *x* y *y*.

Gráficas surf

Las gráficas surf son similares a las gráficas **mesh**, pero **surf** crea una superficie tridimensional colorida en lugar de una mesh. Los colores varían con el valor de **z**.

El comando **surf** toma la misma entrada que **mesh**: una sola entrada (por ejemplo, **surf(z)**, en cuyo caso usa los índices fila y columna como coordenadas *x* y *y*), o tres matrices. La figura 5.29b se generó con los mismos comandos que los usados para generar la figura 5.29a, excepto que **surf** sustituyó **mesh**.

El esquema sombreado para las gráficas de superficie se controla con el comando shading. Por defecto, como en la figura 5.29b, es "plano facetado". El sombreado interpolado puede crear efectos interesantes. La gráfica que se muestra en la figura 5.29c se creó al agregar

```
shading interp
```

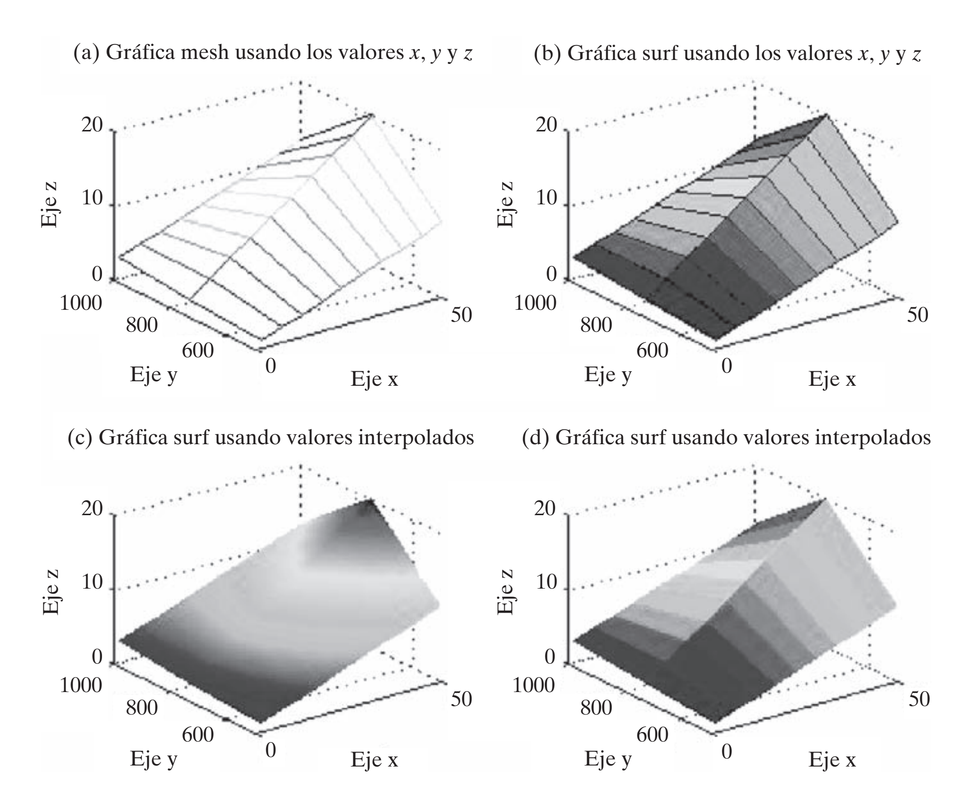


Figura 5.29
Gráficas mesh y surf creadas con tres argumentos de entrada.

a la lista de comandos anterior. El sombreado plano sin retícula se genera cuando se usa

```
shading flat
```

como se muestra en la figura 5.29d.

El esquema de color usado en las gráficas de superficie se puede controlar con la función **colormap**. Por ejemplo,

Idea clave: La función colormap controla los colores usados en las gráficas de superficie.

```
colormap(gray)
```

fuerza una representación en escala de grises para gráficas de superficie. Esto puede ser adecuado si usted usará copias en blanco y negro de sus gráficas. Otros **colormap** disponibles son

autumn	**bone**	**hot**
spring	**colorcube**	**hsv**
summer	**cool**	**pink**
winter	**copper**	**prism**
jet (default)	**flag**	**white**

Use el comando **help** para ver una descripción de las diversas opciones:

```
help colormap
```

Otro ejemplo

Se puede crear una superficie más complicada al calcular los valores de **Z**:

```
x= [-2:0.2:2];
y= [-2:0.2:2];
[X,Y] = meshgrid(x,y);
Z = X.*exp(-X.^2 - Y.^2);
```

En el código precedente, se usa la función **meshgrid** para crear las matrices bidimensionales **X** y **Y** de los vectores unidimensionales **x** y **y**. Entonces se calculan los valores en **Z**. El siguiente código grafica los valores calculados:

```
subplot(2,2,1)
mesh(X,Y,Z)
title('Gráfica mesh'), xlabel('eje x'), ylabel('eje y'),
zlabel('eje z')

subplot(2,2,2)
surf(X,Y,Z)
title('Gráfica de superficie'), xlabel('eje x'), ylabel('eje y'),
zlabel('eje z')
```

Sugerencia

Si en la función **meshgrid** se usa un solo vector, el programa lo interpreta como

```
[X,Y] = meshgrid(x,x)
```

También podría usar la definición del vector como entrada a **meshgrid**:

```
[X,Y] = meshgrid(-2:0.2:2)
```

Ambas líneas de código producirían el mismo resultado que los comandos mencionados en el ejemplo.

Para definir los ejes x y y se pueden usar los vectores **x**, **y** o las matrices **X**, **Y**. La figura 5.30a es una gráfica **mesh** de la función dada, y la figura 5.39b es una gráfica **surf** de la misma función.

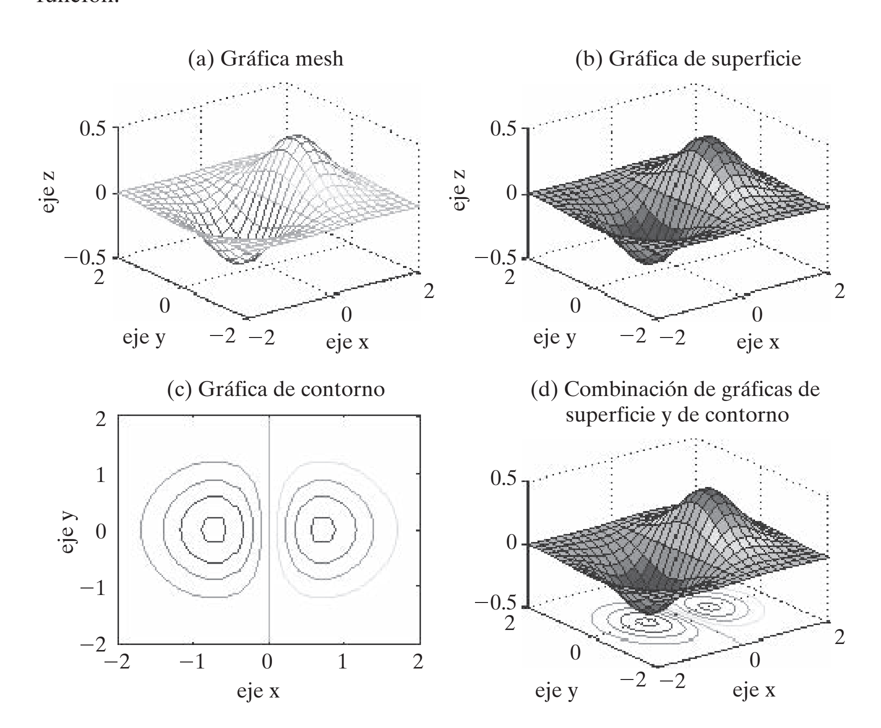


Figura 5.30
Las gráficas de superficie y de contorno son diferentes formas de visualizar los mismos datos.

Gráficas de contorno

Las gráficas de contorno son representaciones bidimensionales de superficies tridimensionales. Para crear la figura 5.30c se usó el comando **contour**, y para crear la figura 5.30d se usó el comando **surfc**:

```
subplot(2,2,3)
contour(X,Y,Z)
xlabel('eje x'), ylabel('eje y'), title('Gráfica de contorno')

subplot(2,2,4)
surfc(X,Y,Z)
xlabel('eje x'), ylabel('eje y')
title('Combinación de gráficas de superficie y de contorno')
```

Gráficas en pseudocolor

Las gráficas en pseudocolor son similares a las gráficas de contorno, excepto que, en lugar de líneas que resaltan un contorno específico, se genera un mapa bidimensional sombreado sobre una retícula. MATLAB incluye una función muestra llamada **peaks** que genera las matrices **x**, **y** y **z** de una interesante superficie que parece una montaña:

```
[x,y,z] = peaks;
```

Con el siguiente código se puede usar esta superficie para demostrar el uso de las gráficas de pseudocolor, que se muestra en la figura 5.31:

```
subplot(2,2,1)
pcolor(x,y,z)
```

La retícula se borra cuando se usa sombreado interpolado:

```
subplot(2,2,2)
pcolor(x,y,z)
shading interp
```

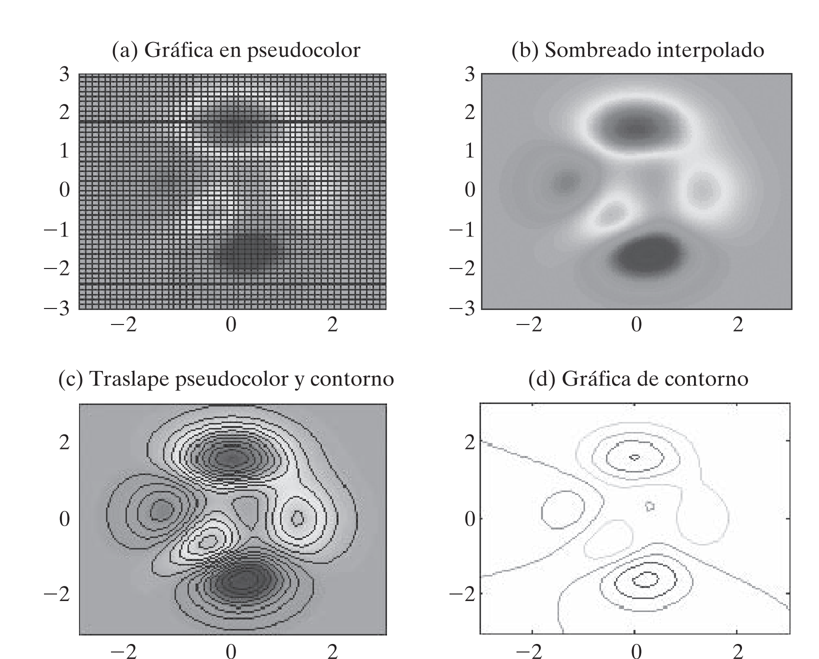


Figura 5.31
En MATLAB están disponibles varias gráficas de contorno.

Puede agregar contornos a la imagen al traslapar una gráfica de contorno:

```
subplot(2,2,3)
pcolor(x,y,z)
shading interp
hold on
contour(x,y,z,20,'k')
```

El número **20** especifica que se dibujan 20 líneas de contorno, y la **'k'** indica que las líneas deben ser negras. Si no se hubiese especificado que las líneas deben ser negras, habrían tenido el mismo color que la gráfica en pseudocolor y habrían desaparecido en la imagen. Finalmente, para comparación, a la figura se agregó una gráfica de contorno simple:

```
subplot(2,2,4)
contour(x,y,z)
```

En la ventana de ayuda se incluyen opciones adicionales para usar todas las funciones de graficación tridimensional.

5.5 EDICIÓN DE GRÁFICAS DESDE LA BARRA DE MENÚ

Idea clave: cuando se edita interactivamente una gráfica, los cambios se pierden si vuelve a correr el programa.

Además de controlar la forma en que se ven sus gráficas con los comandos MATLAB, puede editar una gráfica una vez que la creó. La gráfica de la figura 5.32 se creó con el comando **sphere**, que es una de las muchas funciones muestra, como **peaks**, que se usan para demostrar la graficación.

```
sphere
```

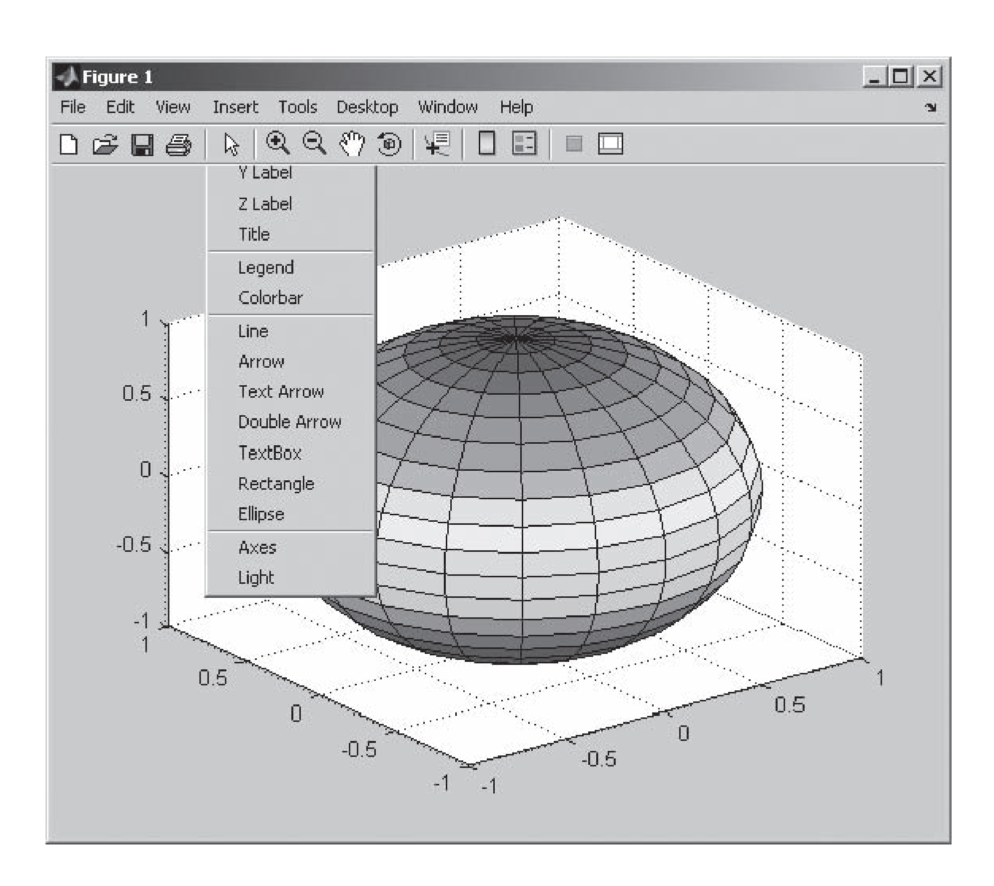


Figura 5.32
Gráfica de una esfera.

En la figura, se seleccionó el **menú Insert**. Note que puede insertar etiquetas, títulos, leyendas, recuadros de texto, etcétera, todos con el uso de este menú. El **menú Tools** le permite cambiar la forma en que se ve la gráfica, al acercarse o alejarse, cambiar la razón de aspecto, etcétera. La figura toolbar (barra de herramientas), bajo el menú del mismo nombre, ofrece iconos que le permiten hacer las mismas cosas.

La gráfica de la figura 5.32 en realidad no parece una esfera; también carece de etiquetas y un título, y puede no ser claro qué significan los colores. Para editar esta gráfica, primero se ajusta la forma:

- Seleccione **Edit → Axis Properties** en el menú toolbar.
- Desde la ventana **Property Editor-Axis**, seleccione **Inspector → Data Aspect Ratio Mode**.
- Establezca el modo a manual. (Véase la figura 5.33.)

De igual modo se agregaron etiquetas, un título y una barra de color (figura 5.34) con la opción **menú Insert** en la barra de menú. Editar su gráfica de esta forma es más interactivo y le permite afinar su apariencia. El único problema con editar interactivamente una figura es que, si corre de nuevo su programa MATLAB, perderá todas sus mejoras.

> **Sugerencia**
>
> Puede forzar a una gráfica a espaciar los datos de manera igualitaria en todos los ejes con el uso del comando **axis equal**. Este enfoque tiene la ventaja de que puede programar **axis equal** en un archivo-m y retener sus mejoras.

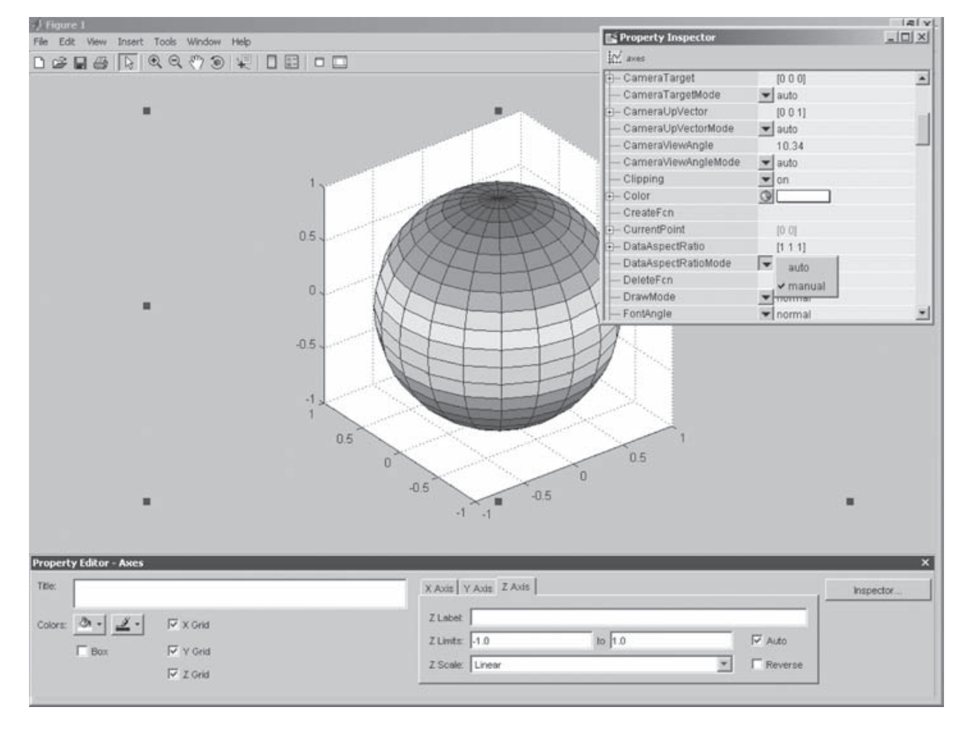


Figura 5.33
MATLAB le permite editar gráficas con los comandos de la barra de herramientas (toolbar).

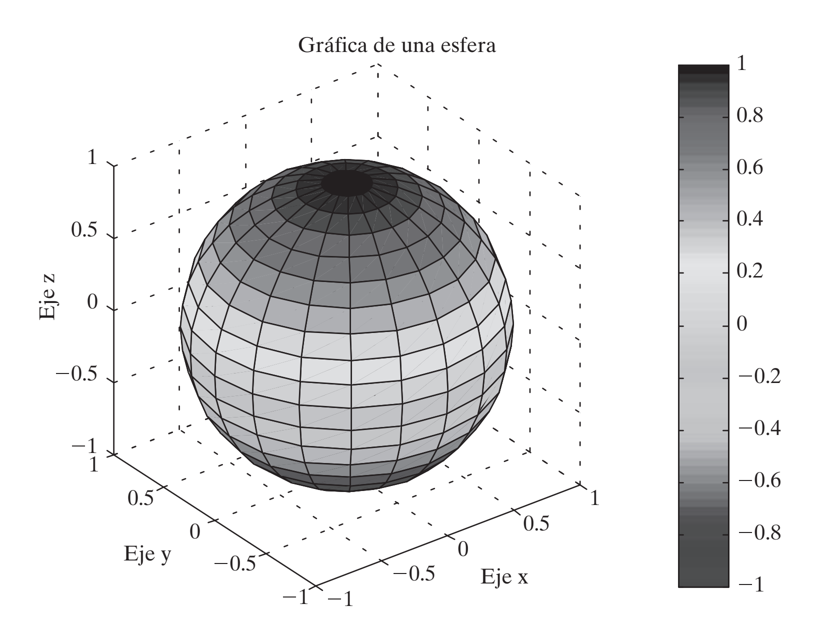


Figura 5.34
Gráfica editada de una esfera.

5.6 CREACIÓN DE GRÁFICAS DESDE LA VENTANA DE TRABAJO

Una gran característica de MATLAB 7 es su habilidad para crear gráficas de manera interactiva desde la ventana del área de trabajo. En esta ventana se selecciona una variable y luego se selecciona el menú desplegable en el **icono graficación** (que se muestra en la figura 5.35). MATLAB mencionará las opciones de graficación que "considera" razonables para los datos almacenados en su variable. Simplemente seleccione la opción adecuada y su gráfica se crea en la **ventana de figura** actual. Si no le gusta alguno de los tipos de gráfica sugeridos, elija **More plots...** en el menú desplegable y se abrirá una nueva ventana con la lista completa de opciones de graficación disponibles para que usted elija. Esto es especialmente útil pues puede sugerir opciones que no se le hayan ocurrido. Por ejemplo, la figura 5.36 es una gráfica de tallo de la matriz **x** resaltada en la figura.

Si quiere graficar más de una variable, resalte la primera y luego mantenga presionada la tecla **Ctrl** y seleccione las variables adicionales. Para anotar sus gráficas, use el proceso de edición interactivo descrito en la sección 5.5. El ambiente interactivo es un recurso rico. Obtendrá mucho si lo explora y experimenta con él.

5.7 CÓMO GUARDAR LAS GRÁFICAS

Existen muchas maneras de guardar las gráficas creadas en MATLAB:

- Si creó la gráfica con código de programación almacenado en un archivo-m, simplemente volver a correr el código recreará la figura.
- También puede guardar la figura desde el menú de archivo, con la opción **Save As...** Se le presentarán varias opciones:
  1. Puede guardar la figura como un archivo **.fig**, que es un formato de archivo específico de MATLAB. Para recuperar la figura, sólo dé doble clic en el nombre del archivo en el directorio actual.

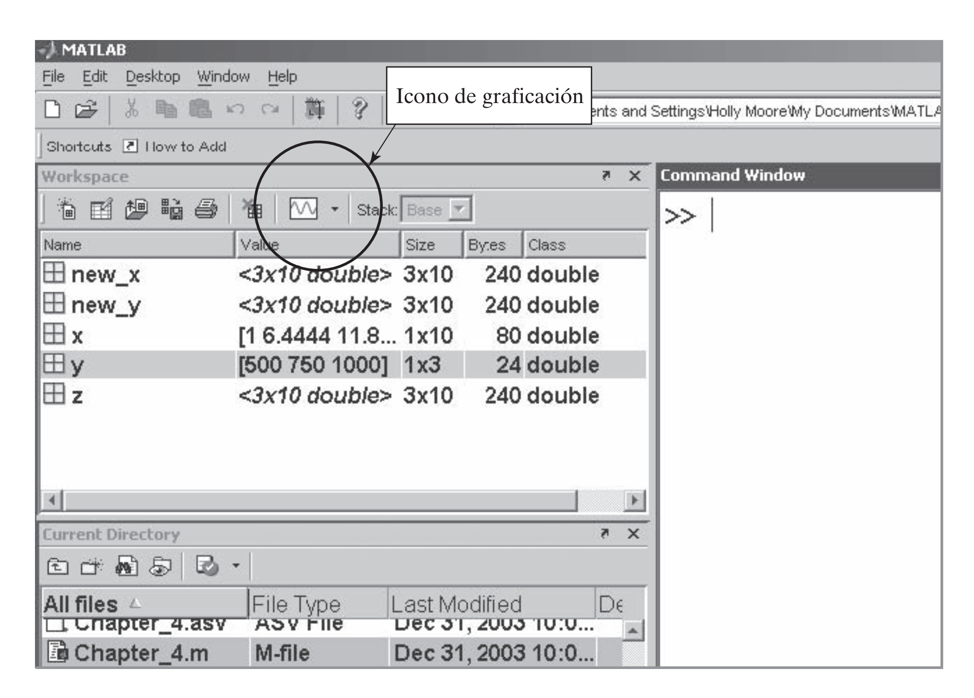


Figura 5.35
Graficación desde la ventana de área de trabajo.

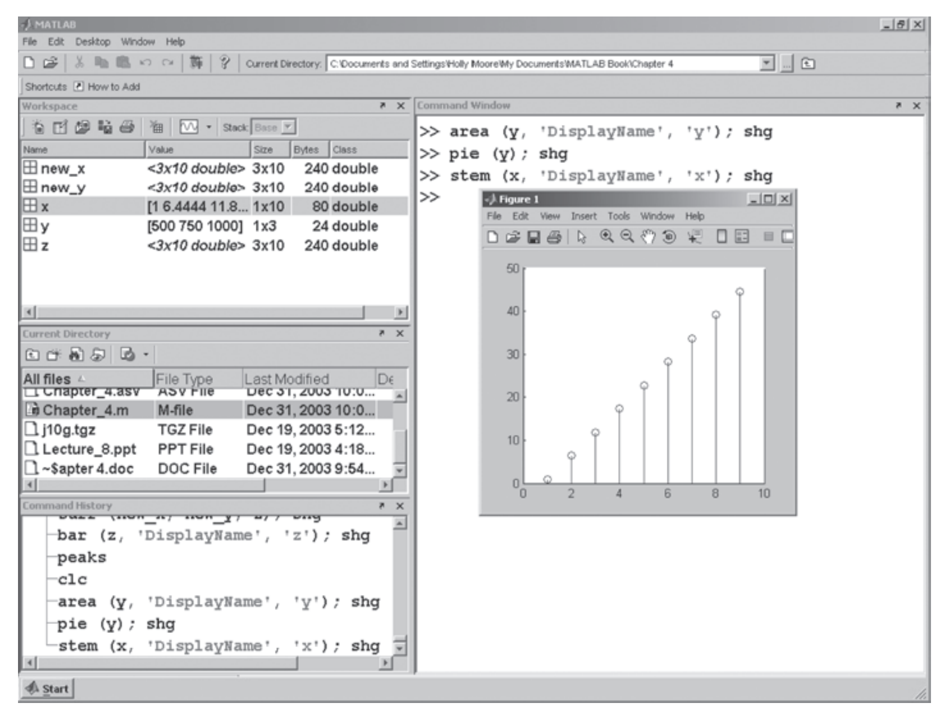


Figura 5.36
Gráfica de tallo creada de manera interactiva desde la ventana del área de trabajo.

2. Puede guardar la figura en diferentes formatos gráficos estándar, como jpeg (**.jpg**) y metarchivo mejorado (**.emf**). Estas versiones de la figura se pueden insertar en otros documentos, como un documento Word. Las figuras en este texto se guardaron como metarchivos mejorados.
3. Puede dar clic derecho en la figura, luego seleccionar **copy** y pegarlo en otro documento.

- Puede usar el menú de archivo para crear un nuevo archivo-m que recreará la figura.

Ejercicio de práctica 5.6

Cree una gráfica de $y = \cos(x)$. Practique guardando el archivo e insertándolo en un documento Word.

RESUMEN

La gráfica usada con más frecuencia en ingeniería es la gráfica *x*-*y*. esta gráfica bidimensional se puede usar para graficar datos o visualizar funciones matemáticas. Sin importar qué represente una gráfica, siempre debe incluir un título y etiquetas de los ejes *x* y *y*. Las etiquetas de los ejes deben ser descriptivas y deben incluir unidades, como ft/s o kJ/kg.

MATLAB incluye opciones extensas para controlar la apariencia de sus gráficas. El usuario puede especificar el color, estilo de línea y estilo de marcador para cada línea en una gráfica. Se puede agregar una retícula a la gráfica y ajustar el rango del eje. Los recuadros de texto y las leyendas se emplean para describir la gráfica. La función subgráfica se usa para dividir la ventana de gráfica en una retícula $m \times n$. Dentro de cada una de estas subventanas, se puede crear y modificar cualquiera de las gráficas MATLAB.

Además de las gráficas *x*-*y*, MATLAB ofrece una diversidad de opciones de graficación, incluidas gráficas polares, gráficas de pastel, gráficas de barras, histogramas y gráficas *x*-*y* con dos ejes *y*. Es posible modificar la escala en las gráficas *x*-*y* para producir gráficas logarítmicas en cualquiera o en ambos ejes *x* y *y*. Los ingenieros usan con frecuencia escala logarítmica para representar datos como una línea recta.

La función **fplot** permite al usuario graficar una función sin definir un vector de valores *x* y *y*. MATLAB elige automáticamente el número adecuado de puntos y espaciamiento para producir una gráfica suave. En la caja de herramientas simbólica hay disponibles capacidad adicional de funciones de graficación.

Las opciones de graficación tridimensionales en MATLAB incluyen una gráfica lineal, algunas gráficas de superficie y gráficas de contorno. La mayoría de las opciones disponibles en la graficación bidimensional también se aplican a estas gráficas tridimensionales. La función **meshgrid** es especialmente útil en la creación de gráficas de superficie tridimensionales.

Las herramientas interactivas permiten al usuario modificar las gráficas existentes. Dichas herramientas están disponibles desde la barra de menú figura. También se pueden crear gráficas con la opción de graficación interactiva de la ventana del área de trabajo. El ambiente interactivo es un recurso rico. Obtendrá mucho si lo explora y experimenta con él.

Las figuras creadas en MATLAB se pueden guardar en varias formas, para editarse más tarde o para insertarse en otros documentos. MATLAB ofrece formatos de archivo propietario que minimizan el espacio de almacenamiento requerido para almacenar figuras y formatos de archivo estándar adecuados para importar hacia otras aplicaciones.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres especiales

Tipo de línea	Indicador	Tipo de punto	Indicador	Color	Indicador
sólida	-	punto	.	azul	b
punteada	:	círculo	o	verde	g
raya-punto	-.	marca x	x	rojo	r
rayada	--	más	+	cian	c
		estrella	*	magenta	m
		cuadrado	s	amarillo	y
		diamante	d	negro	k
		triángulo abajo	v		
		triángulo arriba	^		
		triángulo izquierdo	<		
		triángulo derecho	>		
		pentagrama	p		
		hexagrama	h		

Comandos y funciones

autumn	color de mapa opcional usado en gráficas de superficie
axis	congela la escala del eje actual para gráficas subsecuentes o especifica las dimensiones del eje
axis equal	fuerza el mismo espaciamiento de escala para cada eje
bar	genera una gráfica de barras
bar3	genera una gráfica de barras tridimensional
barh	genera una gráfica de barras horizontal
bar3h	genera una gráfica de barras tridimensional horizontal
bone	mapa de color opcional usado en gráficas de superficie
colorcube	mapa de color opcional usado en gráficas de superficie
colormap	esquema de color usado en gráficas de superficie
comet	dibuja una gráfica *x–y* en una secuencia de pseudoanimación
comet3	dibuja una gráfica lineal tridimensional en una secuencia de pseudoanimación
contour	genera un mapa de contorno de una superficie tridimensional
cool	mapa de color opcional usado en gráficas de superficie
copper	mapa de color opcional usado en gráficas de superficie
figure	abre una nueva ventana de figura
flag	mapa de color opcional usado en gráficas de superficie
fplot	crea una gráfica *x–y* con base en una función
grid	agrega una retícula sólo a la gráfica actual
grid off	desactiva la retícula
grid on	agrega una retícula a la gráfica actual y a todas las subsecuentes en la figura actual
hist	genera un histograma
hold off	instruye a MATLAB a borrar los contenidos de figura antes de agregar nueva información
hold on	instruye a MATLAB a no borrar los contenidos de figura antes de agregar nueva información
hot	mapa de color opcional usado en gráficas de superficie

(Continúa)

Comandos y funciones (continuación)	
`hsv`	mapa de color opcional usado en gráficas de superficie
`jet`	mapa de color por defecto usado en gráficas de superficie
`legend`	agrega una leyenda a una gráfica
`linspace`	crea un vector linealmente espaciado
`loglog`	genera una gráfica *x–y* con ambos ejes escalados logarítmicamente
`mesh`	genera una gráfica mesh de una superficie
`meshgrid`	coloca cada uno de dos vectores en matrices bidimensionales separadas, cuyo tamaño se determina por los vectores fuente
`pause`	pausa la ejecución de un programa hasta que se presiona cualquier tecla
`pcolor`	crea una gráfica de pseudocolor similar a un mapa de contorno
`peaks`	crea una matriz muestra usada para demostrar funciones de graficación
`pie`	genera una gráfica de pastel
`pie3`	genera una gráfica de pastel tridimensional
`pink`	mapa de color opcional usado en gráficas de superficie
`plot`	crea una gráfica *x–y*
`plot3`	genera una gráfica lineal tridimensional
`plotyy`	crea una gráfica con dos ejes *y*
`polar`	crea una gráfica polar
`prism`	mapa de color opcional usado en gráficas de superficie
`semilogx`	genera una gráfica *x–y* con el eje *x* escalado logarítmicamente
`semilogy`	genera una gráfica *x–y* con el eje *y* escalado logarítmicamente
`shading flat`	sombrea una gráfica de superficie con un color por sección de retícula
`shading interp`	sombrea una gráfica de superficie por interpolación
`sphere`	función muestra usada para demostrar graficación
`spring`	mapa de color opcional usado en gráficas de superficie
`subplot`	divide la ventana de gráficas en secciones disponibles para graficación
`summer`	mapa de color opcional usado en gráficas de superficie
`surf`	genera una gráfica de superficie
`surfc`	genera una combinación de gráfica de superficie y de contorno
`text`	agrega un recuadro de texto a una gráfica
`title`	agrega un título a una gráfica
`white`	mapa de color opcional usado en gráficas de superficie
`winter`	mapa de color opcional usado en gráficas de superficie
`xlabel`	agrega una etiqueta al eje *x*
`ylabel`	agrega una etiqueta al eje *y*
`zlabel`	agrega una etiqueta al eje *z*

PROBLEMAS

Gráficas bidimensionales (*x-y*)

5.1 Cree gráficas de las siguientes funciones, desde $x = 0$ hasta 10.

(a) $y = e^x$
(b) $y = \text{sen}(x)$
(c) $y = ax^2 + bx + c$, donde $a = 5, b = 2$ y $c = 4$
(d) $y = \sqrt{x}$

Cada una de sus gráficas debe incluir título, etiqueta del eje *x*, etiqueta del eje *y* y una retícula.

5.2 Grafique el siguiente conjunto de datos:

$$y = [12, 14, 12, 22, 8, 9]$$

Permita que MATLAB use el número de índice de matriz como el parámetro para el eje x.

5.3 Grafique las siguientes funciones en la misma gráfica para valores de x desde $-\pi$ hasta π, y seleccione el espaciamiento para crear una gráfica suave:

$$y_1 = \text{sen}(x)>$$
$$y_2 = \text{sen}(2x)$$
$$y_3 = \text{sen}(3x)$$

(*Sugerencia:* recuerde que la sintaxis MATLAB adecuada para $2x$ es **2*x**.)

5.4 Ajuste la gráfica creada en el problema 5.3 de modo que

- la línea 1 sea roja y rayada.
- la línea 2 sea azul y sólida.
- la línea 3 sea verde y punteada.

No incluya marcadores en ninguna de las gráficas. En general, los marcadores se incluyen sólo en las gráficas de datos medidos, no para valores calculados.

5.5 Ajuste la gráfica creada en el problema 5.4 de modo que el eje x vaya desde -4 hasta $+4$.

- Agregue una leyenda.
- Agregue un recuadro de texto que describa las gráficas.

Graficación *x-y* con proyectiles

Use la siguiente información en los problemas del 5.6 al 5.9:

La distancia que recorre un proyectil cuando se dispara a un ángulo θ es función del tiempo y se puede dividir en distancias horizontal y vertical de acuerdo con las fórmulas

$$\text{Horizontal}(t) = tV_0 \cos(\theta)$$

y

$$\text{Vertical}(t) = tV_0 \,\text{sen}(\theta) - \frac{1}{2} gt^2$$

donde

Horizontal = distancia recorrida en la dirección x,
Vertical = distancia recorrida en la dirección y,
V_0 = velocidad inicial,
g = aceleración debida a gravedad, 9.8 m/s^2,
t = tiempo, s.

5.6 Suponga que el proyectil descrito se dispara con una velocidad inicial de 100 m/s y un ángulo de lanzamiento de $\pi/4$ (45°). Encuentre la distancia recorrida tanto horizontal como verticalmente (en las direcciones x y y) para tiempos desde 0 hasta 20 s.

(a) Grafica distancia horizontal contra tiempo.
(b) En una nueva ventana de figura, grafique distancia vertical contra tiempo (con tiempo en el eje x).

No olvide un título y etiquetas.

5.7 En una nueva ventana de figura, grafique distancia horizontal sobre el eje x y distancia vertical sobre el eje y.

5.8 Calcule tres nuevos vectores para cada una de las distancias vertical (v_1, v_2, v_3) y horizontal (h_1, h_2, h_3) recorridas, y suponga ángulos de lanzamiento de $\pi/2$, $\pi/4$ y $\pi/6$.

- En una nueva ventana de figura, grafique distancia horizontal en el eje x y distancia vertical en el eje y, para los tres casos. (Tendrá tres líneas.)
- Haga una línea sólida, una rayada y una punteada. Agregue una leyenda para identificar cuál línea es cuál.

5.9 Recree la gráfica del problema 5.8. Esta vez, cree una matriz **theta** de los tres ángulos, $\pi/2$, $\pi/4$ y $\pi/6$. Use la función **meshgrid** para crear una malla de **theta** y el vector tiempo (**t**). Luego use las dos nuevas variables en malla que creó para recalcular la distancia vertical (**v**) y la distancia horizontal (**h**) recorridas. Cada uno de sus resultados debe ser una matriz 20×30. Use el comando **plot** para graficar **h** en el eje x y **v** en el eje y.

Uso de subgráficas

5.10 En el problema 5.1, usted creó cuatro gráficas. Combínelas en una figura con cuatro subventanas, con la función **subplot** de MATLAB.

5.11 En los problemas del 5.6 al 5.8, usted creó cuatro gráficas. Combínelas en una figura con cuatro subventanas, con la función **subplot** de MATLAB.

Gráficas polares

5.12 Cree un vector de ángulos desde 0 hasta 2π. Use la función de graficación **polar** para crear gráficas de las funciones que siguen. Recuerde: las gráficas polares esperan el ángulo y el radio como las dos entradas a la función **polar**. Use la función **subplot** para poner sus cuatro gráficas en la misma figura.

(a) $r = \text{sen}^2(\theta) + \cos^2(\theta)$
(b) $r = \text{sen}(\theta)$
(c) $r = e^{\theta/5}$
(d) $r = \text{senh}(\theta)$

5.13 En el ejercicio de práctica 5.3 usted creó algunas formas interesantes en coordenadas polares. Use dichos ejercicios como ayuda para crear las siguientes figuras:

(a) Cree una "flor" con tres pétalos.
(b) Superponga su figura con ocho pétalos adicionales de la mitad del tamaño de los tres originales.
(c) Cree un corazón.
(d) Cree una estrella de seis puntas.
(e) Cree un hexágono.

Gráficas logarítmicas

5.14 Cuando el interés se compone continuamente, la siguiente ecuación representa el crecimiento de sus ahorros:

$$P = P_0 e^{rt}$$

En esta ecuación,

P = saldo actual,
P_0 = saldo inicial,
r = constante de crecimiento, expresada como fracción decimal, y
t = tiempo invertido.

Determine la cantidad en su cuenta al final de cada año si usted invierte \$1000 a 8% (0.08) durante 30 años. (Elabore una tabla.)

Cree una figura con cuatro subgráficas. Grafique tiempo en el eje x y saldo actual P en el eje y.

(a) En el primer cuadrante, grafique t contra P en un sistema coordenado rectangular.
(b) En el segundo cuadrante, grafique t contra P, con escala logarítmica en el eje x.
(c) En el tercer cuadrante, grafique t contra P, con escala logarítmica en el eje y.
(d) En el cuarto cuadrante, grafique t contra P, con escala logarítmica en ambos ejes.

¿Cuál de las cuatro técnicas de graficación considera que muestra mejor los datos?

5.15 De acuerdo con la ley de Moore (una observación hecha en 1965 por Gordon Moore, cofundador de Intel Corporation; véase la figura P5.15), el número de transistores que encajaría por pulgada cuadrada en un circuito integrado semiconductor se duplica aproximadamente cada 18 meses. El año 2005 fue el 40 aniversario de la ley. Durante los últimos 40 años, su proyección se ha satisfecho de manera consistente. En 1965, la entonces tecnología de avanzada permitía 30 transistores por pulgada cuadrada. La ley de Moore dice que la densidad de transistores se puede predecir mediante $d(t) = 30\left(2^{\frac{t}{1.5}}\right)$, donde t se mide en años.



Figura P5.15
Gordon Moore. (Cortesía de Intel Corporation.)

(a) Sea $t = 0$ la representación del año 1965 y $t = 45$ la representación de 2010. Use este modelo para calcular el número predicho de transistores por pulgada cuadrada para los 45 años desde 1965 hasta 2010. Sea t el aumento en incrementos de 1.5 años. Muestre los resultados en una tabla con 2 columnas, una para el año y otra para el número de transistores.
(b) Con la característica **subplot**, grafique los datos en una gráfica lineal x-y, una gráfica x semilog, una gráfica y semilog y una gráfica log-log. Asegúrese de poner título y etiqueta a los ejes.

5.16 Muchos fenómenos físicos se pueden describir mediante la ecuación Arrhenius. Por ejemplo, las constantes de tasa de reacción para reacciones químicas se modelan como

$$k = k_0 e^{(-Q/RT)}$$

donde k_0 = constante con unidades que dependen de la reacción,
Q = energía de activación, kJ/kmol,
R = constante de gas ideal, kJ/kmol K, y
T = temperatura en K.

Para cierta reacción química, los valores de las constantes son

$$Q = 1000 \text{ J/mol},$$
$$k_0 = 10 \text{ s}^{-1}, \text{ y}$$
$$R = 8.314 \text{ J/mol K},$$

para T desde 300 K hasta 1000 K. Encuentre los valores de k. Cree las siguientes dos gráficas de sus datos en una sola ventana de figura:

(a) Grafique T en el eje x y k en el eje y.
(b) Grafique sus resultados como el $\log_{10}$ de k en el eje y y $1/T$ en el eje x.

Gráficas de barras, gráficas de pastel e histogramas

5.17 Sea el vector

$$G = [68, 83, 61, 70, 75, 82, 57, 5, 76, 85, 62, 71, 96, 78, 76, 68, 72, 75, 83, 93]$$

que representa la distribución de calificaciones finales en un curso de ingeniería.

(a) Use MATLAB para ordenar los datos y cree una gráfica de barras de las calificaciones.
(b) Cree un histograma de las calificaciones.

5.18 En la clase de ingeniería mencionada en el problema 5.17 hay

2 A
4 B
8 C
4 D
2 E

(a) Cree una gráfica de pastel de esta distribución. Agregue una leyenda que mencione los nombres de calificación (A, B, C, etcétera).
(b) Use la opción de **menú texto** para agregar un recuadro de texto a cada rebanada de pastel en lugar de una leyenda, y guarde su gráfica modificada como un archivo **.fig**.
(c) Cree una gráfica de pastel tridimensional de los mismos datos. MATLAB 7 tiene problemas con las leyendas para muchas figuras tridimensionales, así que no se sorprenda si su leyenda no se ajusta a la gráfica de pastel.

5.19 En la siguiente tabla se menciona el inventario de cierto tipo de tornillo en un almacén al final de cada mes:

	2004	**2005**
Enero	2345	2343
Febrero	4363	5766
Marzo	3212	4534
Abril	4565	4719
Mayo	8776	3422
Junio	7679	2200
Julio	6532	3454
Agosto	2376	7865
Septiembre	2238	6543
Octubre	4509	4508
Noviembre	5643	2312
Diciembre	1137	4566

Grafique los datos en una gráfica de barras.

5.20 Use la función **randn** para crear 1000 valores en una distribución normal (gaussiana) de números con una media de 70 y una desviación estándar de 3.5. Cree un histograma del conjunto de datos que calculó.

Gráficas con dos ejes *y*

5.21 En la introducción a los problemas del 5.6 al 5.9, aprendió que las ecuaciones para la distancia recorrida por un proyectil como función del tiempo son

$$\text{Horizontal}(t) = tV_0 \cos(\theta)$$

$$\text{Vertical}(t) = tV_0 \,\text{sen}(\theta) - \tfrac{1}{2}gt^2$$

Para tiempo desde 0 hasta 20 s, grafique distancia horizontal contra tiempo y distancia vertical contra tiempo en la misma gráfica, y use ejes *y* separados para cada línea. Suponga un ángulo de lanzamiento de 45 grados (π/4 radianes) y una velocidad inicial de 100 m/s. Suponga también que la aceleración debida a la gravedad, *g*, es 9.8 m/s.

5.22 Si la ecuación que modela la distancia vertical recorrida por un proyectil como función del tiempo es

$$\text{Vertical}(t) = tV_0 \operatorname{sen}(\theta) - 1/2gt^2$$

entonces, del cálculo, la velocidad en la dirección vertical es

$$\text{Velocidad}(t) = V_0 \operatorname{sen}(\theta) - gt$$

Cree un vector t desde 0 hasta 20 s y calcule la posición vertical y la velocidad en la dirección vertical, si supone un ángulo de lanzamiento θ de $\pi/4$ radianes y una velocidad inicial de 100 m/s. Grafique ambas cantidades en la misma gráfica con ejes y separados.

La velocidad debería ser cero en el punto donde el proyectil tiene la mayor altura en la dirección vertical. ¿Su gráfica apoya esta predicción?

5.23 La deformación de muchos metales cambia sus propiedades físicas. En un proceso llamado *trabajo en frío*, el metal se deforma intencionalmente para hacerlo más fuerte. Los siguientes datos tabulan tanto la fortaleza como la ductilidad de un metal que se trabajó en frío a diferentes grados:

Porcentaje trabajo en frío	Fuerza producida, MPa	Ductilidad, %
10	275	43
15	310	30
20	340	23
25	360	17
30	375	12
40	390	7
50	400	4
60	407	3
68	410	2

Grafique estos datos en una sola gráfica x-y con dos ejes y.

Gráficas lineales tridimensionales

5.24 Cree un vector **x** de valores desde 0 hasta 20 π, con un espaciamiento de $\pi/100$. Defina los vectores **y** y **z** como

$$y = x \operatorname{sen}(x)$$

y

$$z = x \cos(x)$$

(a) Cree una gráfica x-y de **x** y **y**.
(b) Cree una gráfica polar de **x** y **y**.
(c) Cree una gráfica lineal tridimensional de **x**, **y** y **z**. No olvide un título y etiquetas.

5.25 Imagine cómo puede ajustar su entrada a **plot3** en el problema 5.24 para crear una gráfica que luzca como un tornado. (Véase la figura P5.25.)

Figura P5.25
Gráfica de tornado.

Gráficas de superficie y contorno tridimensionales

5.26 Cree vectores **x** y **y** desde −5 hasta +5 con un espaciamiento de 0.5. Use la función **meshgrid** para mapear **x** y **y** en dos nuevas matrices bidimensionales llamadas **X** y **Y**. Use sus nuevas matrices para calcular el vector **Z**, con magnitud

$$Z = \operatorname{sen}\left(\sqrt{X^2 + Y^2}\right)$$

(a) Use la función de graficación **mesh** para crear una gráfica tridimensional de **Z**.

(b) Use la función de graficación **surf** para crear una gráfica tridimensional de **Z**. Compare los resultados que obtuvo con una sola entrada (**Z**) con los obtenidos con entradas para las tres dimensiones (**X**, **Y**, **Z**).

(c) Modifique su gráfica de superficie con sombreado interpolado. Intente usar diferentes **colormaps**.

(d) Genere una gráfica de contorno de **Z**.

(e) Genere una combinación de gráficas de superficie y de contorno de **Z**.



CAPÍTULO 6

Funciones definidas por el usuario

Objetivos

Después de leer este capítulo, el alumno será capaz de

- crear y usar sus propias funciones MATLAB con entradas y salidas sencillas y múltiples.
- almacenar y acceder a sus propias funciones en cajas de herramientas.
- crear funciones anónimas.

INTRODUCCIÓN

El lenguaje de programación MATLAB se construye alrededor de funciones. Una *función* es una pieza de código de computación que acepta un argumento de entrada del usuario y produce salida al programa. Las funciones le ayudan a programar eficientemente, lo que le permite evitar reescribir el código de computación para cálculos que se realizan con frecuencia. Por ejemplo, la mayoría de los programas de cómputo contienen una función que calcula el seno de un número. En MATLAB, **sin** es el nombre de la función que se usa para llamar una serie de comandos que realizan los cálculos necesarios. El usuario necesita proporcionar un ángulo y MATLAB regresa un resultado. No es necesario que el programador sepa cómo MATLAB calcula el valor de **sin**(**x**).

6.1 CREACIÓN DE ARCHIVOS-M DE FUNCIÓN

Ya se exploraron muchas de las funciones internas de MATLAB, pero es posible que usted quiera definir sus propias funciones, aquellas que usted usa más comúnmente en su programación. Las funciones definidas por el usuario se almacenan como archivos-m y MATLAB puede acceder a ellas si están almacenadas en el directorio actual.

6.1.1 Sintaxis

Tanto las funciones internas de MATLAB como las definidas por el usuario tienen la misma estructura. Cada una consiste en un nombre, una entrada proporcionada por el usuario y una salida calculada. Por ejemplo, la función

```
cos(x)
```

- se llama **cos**,
- toma la entrada del usuario dentro de paréntesis (en este caso, **x**), y
- calcula un resultado.

El usuario no necesita ver los cálculos realizados, sino que sólo acepta la respuesta. Las funciones definidas por el usuario funcionan de la misma forma. Imagine que usted creó una función llamada **my_function**. Al usar

```
my_function(x)
```

Idea clave: las funciones le permiten programar más eficientemente.

en un programa o desde la ventana de comandos regresaría un resultado, en tanto **x** esté definida y funcione la lógica en la definición de función.

Las funciones definidas por el usuario se crean en archivos-m. Cada una debe comenzar con una línea de definición de función que contenga

- la palabra **function**,
- una variable que defina la salida de función,
- un nombre de función, y
- una variable que se use para el argumento de entrada.

Por ejemplo,

```
function   output =my_function(x)
```

es la primera línea de la función definida por el usuario llamada **my_function**. Requiere un argumento de entrada, que el programa llamará **x**, y calculará un argumento de salida, que el programa llamará **output**. El nombre de función y los nombres de las variables de entrada y salida son arbitrarios y los selecciona el programador. He aquí un ejemplo de una primera línea adecuada para una función llamada calculation (cálculo):

```
function  result = calculation(a)
```

En este caso, el nombre de función es **calculation**, el argumento de entrada se llamará **a** en cualquier cálculo que realice el programa function y la salida se llamará **result**. Aunque se puede usar cualquier nombre MATLAB válido, es buena práctica de programación usar nombres significativos para todas las variables y para nombres de función.

función: pieza de código de computación que acepta una entrada, realiza un cálculo y proporciona una salida

Sugerencia

Los estudiantes se confunden frecuentemente con el uso de la palabra *input* (entrada) cuando se refiere a una función. Aquí se le usa para describir el argumento de entrada, el valor que va adentro de los paréntesis cuando se llama una función. En MATLAB, los argumentos de entrada son diferentes del comando **input**.

He aquí un ejemplo de una función MATLAB muy simple que calcula el valor de un polinomio particular:

```
function output = poly(x)
%Esta función calcula el valor de un polinomio
%de tercer orden
output = 3*x.^3 + 5*x.^2 - 2*x +1;
```

El nombre de la función es **poly**, el argumento de entrada es **x** y la variable de salida se llama **output**.

Antes de poder usar esta función, se debe guardar en el directorio actual. El nombre de archivo debe ser el mismo que el nombre de función con la finalidad de que MATLAB lo encuentre. Todas las convenciones de nomenclatura de MATLAB que se aprendieron para nombrar variables se aplican para nombrar funciones definidas por el usuario. En particular

- El nombre de la función debe comenzar con una letra.
- Puede formarse con letras, números y el guión bajo.
- No se pueden usar nombres reservados.
- Permite cualquier longitud, aunque los nombres largos no son una buena práctica en programación.

Una vez guardado el archivo-m, la función está disponible para usar desde la ventana de comando, desde un archivo-m script o desde otra función. Considere la función **poly** recién creada. Si en la ventana de comando se escribe

Idea clave: los nombres de funciones usan las convenciones de nomenclatura MATLAB estándar para las variables.

```
poly(4)
```

entonces MATLAB responde con

```
ans =
      265
```

Si **a** se hace igual a 4 y se usa **a** como en argumento de entrada, se obtiene el mismo resultado:

```
a = 4;
poly(a)

ans =
      265
```

Si se define un vector, se obtiene un vector de respuestas. Por ende,

```
y = 1:5;
poly(y)
```

produce

```
ans =
      7    41   121   265   491
```

Sugerencia

Mientras crea una función, puede ser útil permitir que los cálculos intermedios se impriman en la ventana de comandos. Sin embargo, una vez que complete su "depuración", asegúrese de que toda su salida se suprime. Si no lo hace, verá información extraña en la ventana de comandos.

Ejercicio de práctica 6.1

Cree funciones MATLAB para evaluar las siguientes funciones matemáticas (asegúrese de seleccionar nombres de función significativos):

1. $y(x) = x^2$
2. $y(x) = e^{1/x}$
3. $y(x) = \text{sen}(x^2)$

Cree funciones MATLAB para las siguientes conversiones de unidades (es posible que necesite consultar un manual o Internet para los factores de conversión adecuados):

4. pulgadas a pies
5. calorías a joules
6. watts a BTU/h
7. metros a millas
8. millas por hora (mph) a pies/s

EJEMPLO 6.1

Conversión entre grados y radianes

Los ingenieros usualmente miden los ángulos en grados, aunque la mayoría de los programas de cómputo y muchas calculadoras requieren que la entrada a las funciones trigonométricas esté en radianes. Escriba y pruebe una función **DR** que cambia grados a radianes y otra función **RD** que cambia radianes a grados. Sus funciones deben tener capacidad de aceptar entrada escalar y matricial.

1. Establezca el problema.
   Crear y poner a prueba dos funciones, DR y RD, para cambiar grados a radianes y radianes a grados (véase la figura 6.1).
2. Describa las entradas y salidas.

   Entrada Un vector de valores grado
   Un vector de valores radián

   Salida Una tabla que convierte grados a radianes
   Una tabla que convierte radianes a grados

3. Desarrolle un ejemplo a mano.

$$\text{grados} = \text{radianes} \times 180/\pi$$
$$\text{radianes} = \text{grados} \times \pi/180$$

Grados a radianes

Grados	Radianes
0	0
30	$30(\pi/180) = \pi/6 = 0.524$
60	$60(\pi/180) = \pi/3 = 1.047$
90	$90(\pi/180) = \pi/2 = 1.571$



Figura 6.1
Las funciones trigonométricas requieren que los ángulos se expresen en radianes.

4. Desarrolle una solución MATLAB.

```
%Ejemplo 6.1
%
clear, clc
%Defina un vector de valores grado
degrees = 0:15:180;
% Llame la función DR y úsela para encontrar radianes
radians = DR(degrees);
%Cree una tabla para usar en la salida
degrees_radians =[degrees;radians]'
```

```
%Defina un vector de valores radián
radians = 0:pi/12:pi;
%Llame la función RD y úsela para encontrar grados
degrees = RD(radians);
radians_degrees = [radians;degrees]'
```

Las funciones llamadas por el programa son

```
function output=DR(x)
%Esta función cambia grados a radianes
output=x*pi/180;
```

y

```
function output=RD(x)
%Esta función cambia radianes a grados
output=x*180/pi;
```

Recuerde que, con la finalidad de que el archivo-m script encuentre las funciones, deben estar en el directorio actual y se deben llamar **DR.m** y **RD.m**. El programa genera los siguientes resultados en la ventana de comandos:

```
degrees_radians =
     0      0.000
    15      0.262
    30      0.524
    45      0.785
    60      1.047
    75      1.309
    90      1.571
   105      1.833
   120      2.094
   135      2.356
   150      2.618
   165      2.880
   180      3.142

radians_degrees =
   0.000       0.000
   0.262      15.000
   0.524      30.000
   0.785      45.000
   1.047      60.000
   1.309      75.000
   1.571      90.000
   1.833     105.000
   2.094     120.000
   2.356     135.000
   2.618     150.000
   2.880     165.000
   3.142     180.000
```

5. Ponga a prueba la solución.
   Compare la solución MATLAB con la solución a mano. Dado que la salida es una tabla, es fácil ver que las conversiones generadas por MATLAB corresponden a las calculadas a mano.

EJEMPLO 6.2



Figura 6.2
Microestructuras típicas de hierro (400x). (De *Metals Handbook*, 9a edición, volumen 1, American Society of Metals, Metals Park, Ohio, 1978.)

Tamaño de grano ASTM

Tal vez no esté acostumbrado a pensar que los metales son cristales, pero lo son. Si observa un trozo pulido de metal bajo un microscopio, la estructura se vuelve clara, como se ve en la figura 6.2. Como puede ver, cada cristal (llamado grano en metalurgia) es de un tamaño y forma diferentes. El tamaño de los granos afecta la fortaleza del metal: cuanto más finos sean los granos, más fuerte será el metal.

Puesto que es difícil determinar un tamaño de grano "promedio", la ASTM (anteriormente conocida como la American Society for Testing and Materials, pero ahora conocida sólo por sus siglas en inglés) desarrolló una técnica estándar. Una muestra de metal se examina bajo un microscopio a una amplificación de 100, y se cuenta el número de granos en una pulgada cuadrada. La ecuación relevante es

$$N = 2^{n-1}$$

donde n es el tamaño de grano ASTM y N es el número de granos por pulgada cuadrada a 100×. La ecuación se puede resolver para n, lo que produce

$$n = \frac{(\log(N) + \log(2))}{\log(2)}$$

Esta ecuación no es difícil de usar, pero es complicada. En vez de ello, cree una función MATLAB llamada **grain_size** (tamaño de grano).

1. Establezca el problema.
   Crear y poner a prueba una función llamada **grain_size** para determinar el tamaño de grano ASTM de un trozo de metal.
2. Describa las entradas y salidas.
   Para poner a prueba la función, necesitará elegir un número arbitrario de granos. Por ejemplo:

   Entrada 16 granos por pulgada cuadrada a 100×

   Salida tamaño de grano ASTM

3. Desarrollo un ejemplo a mano.

$$n = \frac{(\log(N) + \log(2))}{\log(2)}$$

$$n = \frac{(\log(16) + \log(2))}{\log(2)} = 5$$

4. Desarrolle una solución MATLAB.
   La función, que se crea en un archivo-m separado, es

```
function output = grain_size(N)
%Calcula el tamaño de grano ASTM n
output = (log10(N) + log10(2))./log10(2);
```

que se guardó como **grain_size.m** en el directorio actual. Para usar esta función, se le puede llamar desde la ventana de comandos:

```
grain_size(16)

ans =
       5
```

5. Ponga a prueba la solución.
   La solución MATLAB es la misma que la solución a mano. Puede ser interesante ver cómo el tamaño de grano ASTM varía con el número de granos por pulgada cuadrada. Se podría usar la función con un arreglo de valores y graficar los resultados en la figura 6.3.

```
%Ejemplo 6.2
%Tamaño de grano ASTM

N = 1:100;
n = grain_size(N);
plot(N,n)
title('Tamaño de grano ASTM')
xlabel('Número de granos por pulgada cuadrada a 100x')
ylabel('Tamaño de grano ASTM')
grid
```

Como se esperaba, el tamaño de grano aumenta conforme aumenta el número de granos por pulgada cuadrada.

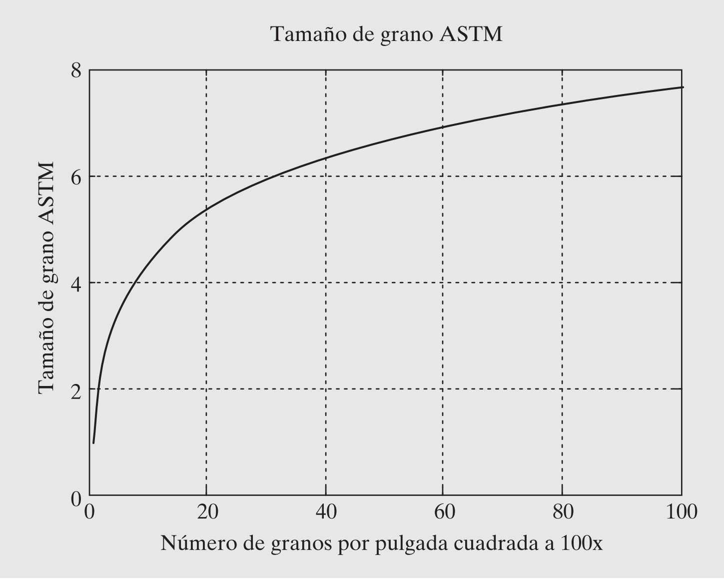


Figura 6.3
La gráfica del comportamiento de una función es una buena forma de ayudar a determinar si la programó correctamente.

Idea clave: los comentarios de función se despliegan cuando se usa la característica help.

6.1.2 Comentarios

Como con cualquier programa de cómputo, debe comentar libremente su código de modo que sea más fácil de seguir. Sin embargo, en una función MATLAB, los comentarios en la línea inmediatamente siguiente a la primera línea tienen un papel especial. Dichas líneas se recuperan cuando se solicita la función **help** en la ventana de comandos. Considere, por ejemplo, la siguiente función:

```
function results = f(x)
%Esta función convierte segundos a minutos
results = x./60;
```

Cuando se solicita la función **help** desde la ventana de comandos, se regresa la línea de comentario. Por ende,

```
help f
```

regresa

```
This function converts seconds to minutes
```

6.1.3 Funciones con entradas y salidas múltiples

Tal como las funciones MATLAB predefinidas pueden requerir múltiples entradas y pueden regresar múltiples salidas, se pueden escribir funciones más complicadas definidas por el usuario. Recuerde, por ejemplo, la función remainder (residuo). Esta función predefinida calcula el residuo en un problema de división y requiere que el usuario ingrese el dividendo y el divisor. Para el problema $\frac{5}{3}$, la sintaxis correcta es

```
rem(5,3)
```

que produce

```
ans =
      2
```

De manera similar, se podría escribir una función definida por el usuario para multiplicar dos vectores:

```
function output=g(x,y)
% Esta función multiplica x y y
% x y y deben ser matrices del mismo tamaño
a = x .*y;
output=a;
```

Cuando **x** y **y** se definen en la ventana de comandos y se llama la función **g**, se regresa un vector de valores de salida:

```
x=1:5;
y=5:9;
g(x,y)
ans =
  5   12   21   32   45
```

Puede usar las líneas de comentario para hacer que los usuarios sepan qué tipo de entrada se requiere y para describir la función. En este ejemplo se realizó un cálculo intermedio (**a**), pero la única salida de esta función es la variable que se llamó **output**. Esta salida puede ser una matriz que contenga una diversidad de números, pero aún así es sólo una variable.

También puede crear funciones que regresen más de una variable de salida. Muchas de las funciones MATLAB predefinidas regresan más de un resultado, por ejemplo, **max** regresa tanto el valor máximo en una matriz como el número de elemento en el que ocurre el máximo. Para lograr el mismo resultado en una función definida por el usuario, haga la salida una matriz de respuestas en lugar de una sola variable, como lo hace la siguiente función:

```
function    [dist, vel, accel] = motion(t)
% Esta función calcula la distancia, velocidad y
% aceleración de un automóvil para un valor dado de t
accel = 0.5 .*t;
vel = accel .* t;
dist = vel.*t;
```

Una vez que la guarde como **motion** en el directorio actual, puede usar la función para encontrar valores de **distance**, **velocity** y **acceleration** en tiempos específicos:

```
[distance, velocity, acceleration] = motion(10)

distance =
      500
velocity =
      50
acceleration =
      5
```

Si llama la función **motion** sin especificar las tres salidas, sólo se regresará la primera salida:

```
motion(10)
ans =
      500
```

Recuerde: todas las variables en MATLAB son matrices, así que es importante usar el operador **.*** en el ejemplo anterior, que especifica la multiplicación elemento por elemento. Por ejemplo, usar un vector de valores de tiempo desde 0 hasta 30 en la función **motion**

```
time = 0:10:30;
[distance, velocity, acceleration] = motion(time)
```

regresa tres vectores de respuestas:

```
distance =
       0   500   4000   13500
velocity =
       0   50    200    450
acceleration =
       0   5     10     15
```

Es más fácil ver los resultados si agrupa los vectores, como en

```
results =[time',distance',velocity',acceleration']
```

lo que regresa

```
results =
           0         0            0            0
          10         500          50           5
          20         4000         200          10
          30         13500        450          15
```

Puesto que **time**, **distance**, **velocity** y **acceleration** fueron vectores fila, se usó el operador transpuesto para hacerlos columnas.

Ejercicio de práctica 6.2

Si supone que las dimensiones de la matriz concuerdan, cree y ponga a prueba funciones MATLAB para evaluar las siguientes funciones matemáticas simples con vectores de entrada múltiple y un vector de salida sencilla:

1. $z(x, y) = x + y$
2. $z(a, b, c) = ab^c$
3. $z(w, x, y) = we^{(x/y)}$
4. $z(p, t) = p/\text{sen}(t)$

Si supone que las dimensiones de la matriz concuerdan, cree y ponga a prueba funciones MATLAB para evaluar las siguientes funciones matemáticas simples con un vector de entrada sencilla y vectores de salida múltiple:

5. $f(x) = \cos(x)$
   $f(x) = \text{sen}(x)$
6. $f(x) = 5x^2 + 2$
   $f(x) = \sqrt{5x^2 + 2}$
7. $f(x) = \exp(x)$
   $f(x) = \ln(x)$

Si supone que las dimensiones de la matriz concuerdan, cree y ponga a prueba funciones MATLAB para evaluar las siguientes funciones matemáticas simples con vectores de entrada múltiple y vectores de salida múltiple:

8. $f(x, y) = x + y$
   $f(x, y) = x - y$
9. $f(x, y) = ye^x$
   $f(x, y) = xe^y$

EJEMPLO 6.3

Cómo el tamaño de grano afecta la fortaleza del metal: Una función con tres entradas

Los metales compuestos de pequeños cristales son más fuertes que los metales compuestos de menos cristales grandes. Una fórmula que relaciona la resistencia a la compresión (la cantidad de tensión a la que el metal comienza a deformarse permanentemente) con el diámetro de grano promedio se llama ecuación Hall-Petch:

$$\sigma = \sigma_0 + Kd^{-1/2}$$

donde los símbolos σ_0 y K representan constantes que son diferentes para cada metal.

Cree una función llamada **HP** que requiera tres entradas (σ_0, K y d) y calcule el valor de la resistencia a la compresión. Llame esta función desde un programa MATLAB que proporcione valores de σ_0 y K, y luego grafique el valor de la resistencia a la compresión para valores de d desde 0.1 hasta 10 mm.

1. Establezca el problema.
   Crear una función llamada HP que determine la resistencia a la compresión de un trozo de metal, con el uso de la ecuación Hall-Petch. Usar la función para crear una gráfica de resistencia a la compresión contra diámetro de grano.
2. Describa las entradas y salidas.

   Entrada $K = 9600\ \text{psi}/\sqrt{\text{mm}}$
   $\sigma_0 = 12{,}000\ \text{psi}$
   $d = 0.1\ \text{a}\ 10\ \text{mm}$

   Salida Gráfica de resistencia a la compresión contra diámetro
3. Desarrolle un ejemplo a mano.
   La ecuación Hall-Petch es

   $$\sigma = \sigma_0 + Kd^{-1/2}$$

   Al sustituir los valores de 12,000 psi y 9600 psi/$\sqrt{\text{mm}}$ para σ_0 y K, respectivamente, se obtiene

   $$\sigma = 12{,}000 + 9600d^{-1/2}$$

   Para $d = 1$ mm,

   $$\sigma = 12{,}000 + 9600 = 21{,}600$$

4. Desarrolle una solución MATLAB.
   La función deseada, que se crea en un archivo-m separado, es

```
function output = HP(sigma0,K,d)
%Ecuación Hall-Petch para determinar la resistencia a
%la compresión de metales
output = sigma0 + K*d.^(-0.5);
```

   y se guardó como **HP.m** en el directorio actual:

```
%Ejemplo 6.3
clear,clc
format compact
s0=12000
K=9600
%Defina los valores de diámetro de grano
d=0.1:0.1:10;
yield=HP(s0,K,d);
%Grafique los resultados
figure(1)
```

```
plot(d,yield)
title('Resistencias a la compresión encontradas con la ecuación
Hall-Petch')
xlabel('Diámetro, mm')
ylabel('Resistencia a la compresión, psi')
```

El programa generó la gráfica que se muestra en la figura 6.4.

5. Ponga a prueba la solución.
   Se puede usar la gráfica para comparar los resultados con la solución a mano.

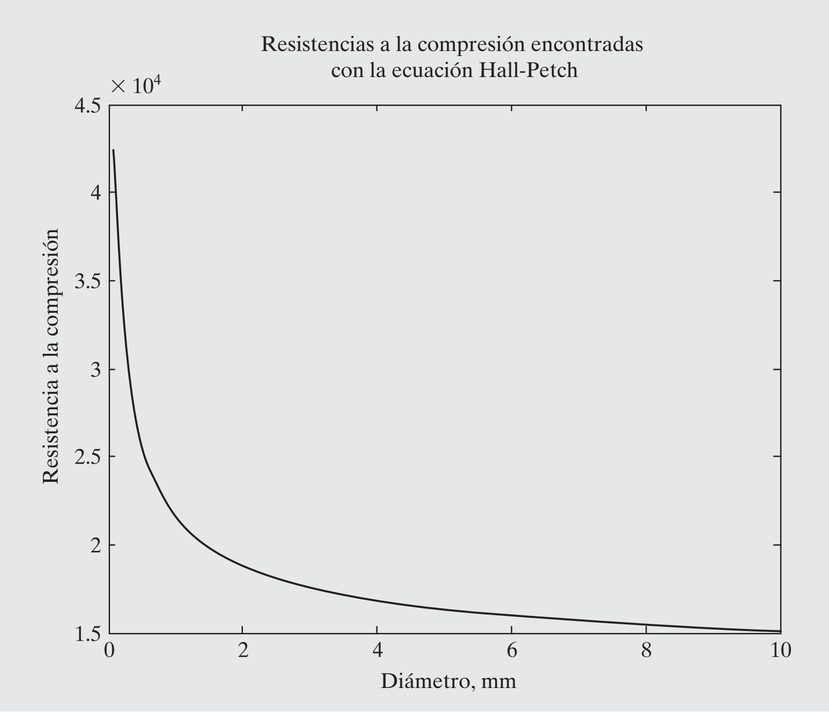


Figura 6.4
Resistencias a la compresión predichas con la ecuación Hall-Petch.

EJEMPLO 6.4

Energía cinética: una función con dos entradas

La energía cinética de un objeto en movimiento (figura 6.5) es

$$\text{KE} = \tfrac{1}{2}mv^2.$$

Cree y ponga a prueba una función llamada KE para encontrar la energía cinética de un automóvil en movimiento si se conocen la masa *m* y la velocidad *v* del vehículo.

1. Establezca el problema.
   Crear una función llamada KE para encontrar la energía cinética de un automóvil.

2. Describa las entradas y salidas.

   Entrada Masa del automóvil, en kilogramos
   Velocidad del automóvil, en m/s

   Salida Energía cinética, en joules

3. Desarrolle un ejemplo a mano.
   Si la masa es 1000 kg y la velocidad es 25 m/s, entonces

$$\text{KE} = \tfrac{1}{2} \times 1000 \text{ kg} \times (25 \text{ m/s})^2 = 312{,}500 \text{ J} = 312.5 \text{ kJ}$$

4. Desarrolle una solución MATLAB.

```
function   output =ke(m,v)
output = 1/2*m*v.^2;
```

5. Ponga a prueba la solución.

```
v = 25;
m = 1000;
ke(m,v)

ans =
      312500
```

   Este resultado coincide con el ejemplo a mano, lo que confirma que la función funciona correctamente y ahora se puede usar en un programa MATLAB más grande.



Figura 6.5
Los autos de carreras almacenan una cantidad significativa de energía cinética.

6.1.4 Funciones sin entrada o salida

Aunque la mayoría de las funciones necesitan al menos una entrada y regresan al menos un valor de salida, en algunas situaciones no se requieren ni entradas ni salidas. Por ejemplo, considere esta función, que dibuja una estrella en coordenadas polares:

```
function [] = star( )
theta = pi/2:0.8*pi:4.8*pi;
r=ones(1,6);
polar(theta,r)
```

Los corchetes de la primera línea indican que la salida de la función es una matriz vacía (es decir: no se regresa valor). Los paréntesis vacíos dicen que no se espera entrada. Si desde la ventana de comandos usted escribe

```
star
```

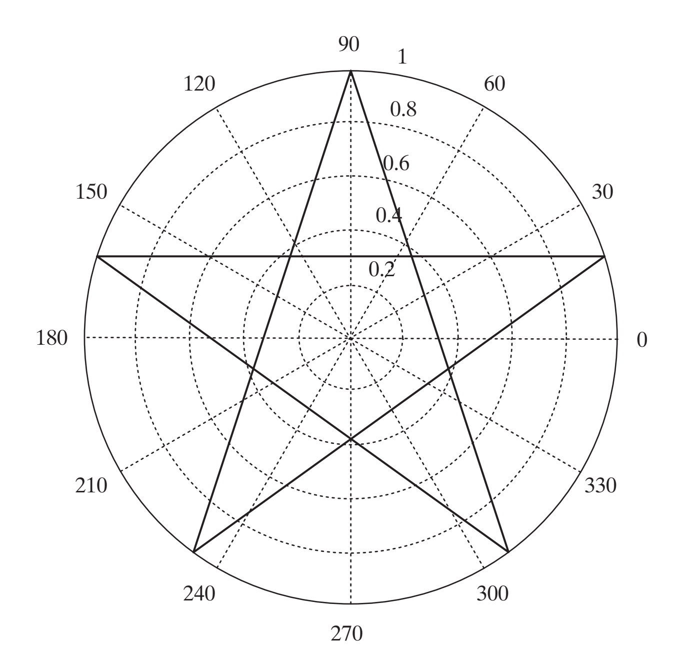


Figura 6.6
La función **star** definida por el usuario no requiere entrada y no produce valores de salida, pero dibuja una estrella en coordenadas polares.

entonces no regresa valores, sino que se abre una ventana de figura que muestra una estrella dibujada en coordenadas polares. (Véase la figura 6.6.)

Sugerencia

Es posible que se pregunte si la función **star** realmente es un ejemplo de una función que no regresa una salida; después de todo, dibuja una estrella. Pero la salida de una función se define como un *valor* que se regresa cuando usted llama la función. Si se pide a MATLAB realizar el cálculo

A = star

se genera un enunciado de error, ¡porque la función **star** no regresa nada! Por ende, no hay nada con lo que se pueda igualar a A.

Idea clave: no todas las funciones requieren una entrada.

Existen muchas funciones internas MATLAB que no requieren entrada alguna. Por ejemplo,

```
A=clock
```

regresa la hora actual:

```
A =
  1.0e+003 *
  Columns 1 through 4
    2.0050    0.0030    0.0200    0.0150
  Columns 5 through 6
    0.0250    0.0277
```

Además,

```
A=pi
```

regresa el valor de la constante matemática π:

```
A =
    3.1416
```

Sin embargo, si se intenta establecer la función MATLAB **tic** igual a un nombre de variable, se genera un enunciado de error porque **tic** no regresa un valor de salida:

```
A=tic
??? Error using ==> tic
Too many output arguments.
```

(La función **tic** inicia un cronómetro para su uso posterior en la función **toc**.)

6.1.5 Determinación del número de argumentos de entrada y salida

En alguna ocasión usted querrá conocer el número de argumentos de entrada o valores de salida asociados con una función. MATLAB proporciona dos funciones internas para este propósito.

La función **nargin** determina el número de argumentos de entrada o en una función definida por el usuario o en una función interna. El nombre de la función se debe especificar como una cadena, como, por ejemplo, en

Idea clave: usar las funciones nargin o nargout es útil en funciones de programación con entradas y salidas variables.

```
nargin('sin')
ans =
   1
```

La función residuo, **rem**, requiere dos entradas; por tanto,

```
nargin('rem')
ans =
   2
```

Cuando **nargin** se usa dentro de una función definida por el usuario, determina cuántos argumentos de entrada se ingresaron en realidad. Esto permite a una función tener un número variable de entradas. Recuerde las funciones de graficación como **surf**. Cuando **surf** tiene una sola entrada matriz, se crea una gráfica, que usa los números de índice de matriz como las coordenadas *x* y *y*. cuando existen tres entradas, *x*, *y* y *z*, la gráfica se basa en los valores *x* y *y* especificados. La función **nargin** permite al programador determinar cómo crear la gráfica, con base en el número de entradas.

La función **surf** es un ejemplo de una función con un número variable de entradas. Si se usa **nargin** desde la ventana de comandos para determinar el número de entradas declaradas, no hay una respuesta correcta. La función **nargin** regresa un número negativo para hacer saber que es posible un número variable de entradas:

```
nargin('surf')
ans =
   -1
```

La función **nargout** es similar a **nargin**, pero determina el número de salidas de una función:

```
nargout('sin')
ans =
   1
```

El número de salidas se determina mediante cuántas matrices se regresan, no cuántos valores hay en la matriz. Se sabe que **size** regresa el número de filas y columnas en una matriz, así que se puede esperar que **nargout** regresa 2 cuando se aplica a size. Sin embargo,

```
nargout('size')
ans =
    1
```

regresa sólo una matriz, que tiene sólo dos elementos, como por ejemplo, en

```
x=1:10;
size(x)
ans =
  1   10
```

Un ejemplo de una función con salidas múltiples es **max**:

```
nargout('max')
ans =
     3
```

Cuando se usa dentro de una función definida por el usuario, **nargout** determina cuántas salidas solicitó el usuario. Considere este ejemplo, en el que se reescribió la función de la sección 6.1.4 para crear una estrella:

```
function A=star1( )
theta = pi/2:0.8*pi:4.8*pi;
r=ones(1,6);
polar(theta,r)
if nargout==1
    A='Twinkle twinkle little star';
end
```

Si se usa **nargout** desde la ventana de comando, como en

```
nargout('star1')
ans =
   1
```

MATLAB indica que se especificó una salida. No obstante, si se llama la función simplemente como

```
star1
```

nada regresa a la ventana de comandos, aunque se dibuje la gráfica. Si se llama la función al igualarla a una variable, como en

```
x=star1
x =
Twinkle twinkle little star
```

se regresa un valor para **x**, con base en el enunciado **if** incrustado en la función, que usó **nargout** para determinar el número de valores de salida.

6.1.6 Variables locales

Las variables que se usan en los archivos-m de función se conocen como *variables locales*. La única forma en que una función puede comunicarse con el área de trabajo es a través de los argumentos de entrada y la salida que regresa. Cualesquiera variables definidas dentro de la

función existen sólo para uso de la función. Por ejemplo, considere la función **g** descrita anteriormente:

```
function output=g(x,y)
% Esta función multiplica x y y
% x y y deben ser matrices del mismo tamaño
a = x .*y;
output=a;
```

Las variables **a**, **x**, **y** y **output** son variables locales. Se pueden usar para cálculos adicionales dentro de la función **g**, pero no se almacenan en el área de trabajo. Para confirmar esto, limpie el área de trabajo y la ventana de comandos y luego llame la función **g**:

variable local: una variable que sólo tiene significado dentro de un programa o función

```
clear, clc
g(10,20)
```

La función regresa

```
g(10,20)
ans =
      200
```

Note que la única variable almacenada en la ventana del área de trabajo es **ans**, que se caracteriza del modo siguiente:

Nombre	Valor	Tamaño	Bytes	Clase
ans	200	1×1	8	double array

Tal como los cálculos realizados en la ventana de comandos o desde un archivo-m script no pueden tener acceso a variables definidas en funciones, las funciones no pueden tener acceso a las variables definidas en el área de trabajo. Esto significa que las funciones deben estar completamente autocontenidas: la única forma en que pueden obtener información de su programa es a través de los argumentos de entrada, y la única forma en que pueden entregar información es a través de la salida de la función.

Considere una función escrita para encontrar la distancia que un objeto cae debido a la gravedad:

```
function  result = distance(t)
%Esta función calcula la distancia que un objeto en caída
%recorre debido a la gravedad
g = 9.8      %metros por segundo al cuadrado
result = 1/2*g*t.^2;
```

El valor de **g** debe incluirse *adentro* de la función. No importa si **g** se usa o no en el programa principal. Cómo se defina **g** está oculto a la función distance, a menos que **g** se especifique adentro del programa.

Desde luego, también podría pasar el valor de **g** a la función como un argumento de entrada:

```
function  result = distance(g,t)
%Esta función calcula la distancia que un objeto en caída
%recorre debido a la gravedad
result = 1/2*g*t.^2;
```

Sugerencia

Para referirlas, se pueden usar los mismos nombres de matriz tanto en una función como en el programa. Sin embargo, no *tienen* que ser los mismos. Dado que los nombres de variable son locales a la función o al programa que llama la función, las variables están completamente separadas. Como programador principiante, debería usar diferentes nombres de variable en sus funciones y sus programas, sólo para que no se confunda *usted mismo*.

6.1.7 Variables globales

Idea clave: por lo general es mala idea definir variables globales.

A diferencia de las variables locales, las variables globales están disponibles para todas las partes de un programa de cómputo. En general, ***es mala idea*** definir variables globales. Sin embargo, MATLAB protege al usuario del uso no previsto de una variable global al requerir que se identifique tanto en el ambiente de la ventana de comandos (o en un archivo-m script) como en la función que la usará.

Considere la función distance una vez más:

variable global: variable que está disponible a partir de programas múltiples

```
function  result = distance(t)
%Esta función calcula la distancia que un objeto en caída
%recorre debido a la gravedad
global G
result = 1/2*G*t.^2;
```

El comando **global** alerta a la función para que busque en el área de trabajo el valor de **G**. **G** también se debe definir en la ventana de comandos (o archivo-m script) como una variable global:

```
global G
G=9.8;
```

Este enfoque le permite cambiar el valor de **G** sin necesidad de redefinir la función distance o proporcionar el valor de **G** como un argumento de entrada a la función distance.

Sugerencia

Como cuestión de estilo, siempre ponga los nombres de las variables globales en mayúsculas. A MATLAB no le importa, pero es más fácil identificar las variables globales si usa una convención de nomenclatura consistente.

Sugerencia

Puede parecer buena idea usar variables globales porque ellas pueden simplificar sus programas. No obstante, considere este ejemplo de uso de variables globales en su vida diaria: sería más fácil comprar un libro en una tienda en línea si publicara la información de su tarjeta de crédito en un sitio donde cualquier vendedor sólo tuviera que buscarla. Entonces el librero no tendría que pedirle que escriba su número. Sin embargo, esto podría producir algunas consecuencias no previstas (¡como que otra persona use su tarjeta de crédito sin su permiso o conocimiento!). Cuando usted crea una variable global, queda disponible a otras funciones y dichas funciones la pueden cambiar, lo que a veces conduce a consecuencias no previstas.

6.1.8 Acceso a código de archivo-m

Las funciones proporcionadas con MATLAB son de dos tipos. Un tipo es interno y el código no es accesible para que el usuario lo revise. El otro tipo consiste en archivos-m, que se almacenan en cajas de herramientas proporcionadas con el programa. Estos archivos-m (o los archivos-m que uno escribe) se pueden ver con el comando **type**. Por ejemplo, la función **sphere** crea una representación tridimensional de una esfera; por tanto,

```
type sphere
```

o

```
type('sphere')
```

regresa los contenidos del archivo **sphere.m**:

```
function [xx,yy,zz] = sphere(varargin)
%SPHERE genera una esfera.
%   [X,Y,Z] = SPHERE(N) genera tres matrices
%   (N+1)-por-(N+1) de modo que SURF(X,Y,Z) produzca una
%   esfera unitaria.
%
%   [X,Y,Z] = SPHERE usa N = 20.
%
%   SPHERE(N) y SPHERE sola grafican la esfera como
%   una SUPERFICIE y no regresan nada.
%
%   SPHERE(AX,...) grafica en AX en lugar de GCA.
%
%   Vea también ELLIPSOID, CYLINDER.
%   Clay M. Thompson 4-24-91, CBM 8-21-92.
%   Copyright 1984-2002 The MathWorks, Inc.
%   $Revision: 5.8.4.1 $  $Date: 2002/09/26 01:55:25 $
% Posible división (parse) de entrada Axes
error(nargchk(0,2,nargin));
[cax,args,nargs] = axescheck(varargin{:});
n = 20;
if nargs > 0, n = args{1}; end
% -pi <= theta <= pi es un vector fila.
% -pi/2 <= phi <= pi/2 es un vector columna.
theta = (-n:2:n)/n*pi;
phi = (-n:2:n)'/n*pi/2;
cosphi = cos(phi); cosphi(1) = 0; cosphi(n+1) = 0;
sintheta = sin(theta); sintheta(1) = 0; sintheta(n+1) = 0;
x = cosphi*cos(theta);
y = cosphi*sintheta;
z = sin(phi)*ones(1,n+1);
if nargout == 0
    cax = newplot(cax);
    surf(x,y,z,'parent',cax)
else
    xx = x; yy = y; zz = z;
end
```

Sugerencia

Note que la función **sphere** usa **varargin** para indicar que aceptará un número variable de argumentos de entrada. La función también utiliza las funciones **nargin** y **nargout**. Estudiar esta función le puede dar ideas acerca de cómo programar sus propios archivos-m de función.

6.2 CREACIÓN DE SU PROPIA CAJA DE HERRAMIENTAS DE FUNCIONES

Idea clave: agrupe sus funciones en cajas de herramientas.

Cuando llama una función en MATLAB, el programa busca primero en el directorio actual para ver si la función está definida. Si no puede encontrar la función ahí, comienza a recorrer una ruta de búsqueda predefinida en busca de un archivo con el nombre de la función. Para ver la ruta que el programa sigue conforme busca los archivos, seleccione

```
File → Set Path
```

de la barra de menú o escriba

```
pathtool
```

en la ventana de comandos (figura 6.7).

Conforme cree más y más funciones para usar en su programación, querrá modificar la ruta para buscar en un directorio donde haya almacenado sus propias herramientas personales. Por ejemplo, suponga que almacenó las funciones grados a radianes y radianes a grados creadas en el ejemplo 6.1 en un directorio llamado **My_functions**.

Puede agregar este directorio (carpeta) a la ruta al seleccionar **Add Folder** de la lista de botones de opción en la ventana de diálogo Set Path (establecer ruta), como se muestra en la figura 6.7. Se le solicitará proporcionar la ubicación de la carpeta o navegar para encontrarla, como se muestra en la figura 6.8.

Ahora MATLAB busca las definiciones de función primero en el directorio actual y luego recorre la ruta de búsqueda modificada, como se muestra en la figura 6.9.

Una vez que agregue una carpeta a la ruta, el cambio se aplica sólo a la sesión actual de MATLAB, a menos que guarde sus cambios de manera permanente. Está claro que nunca

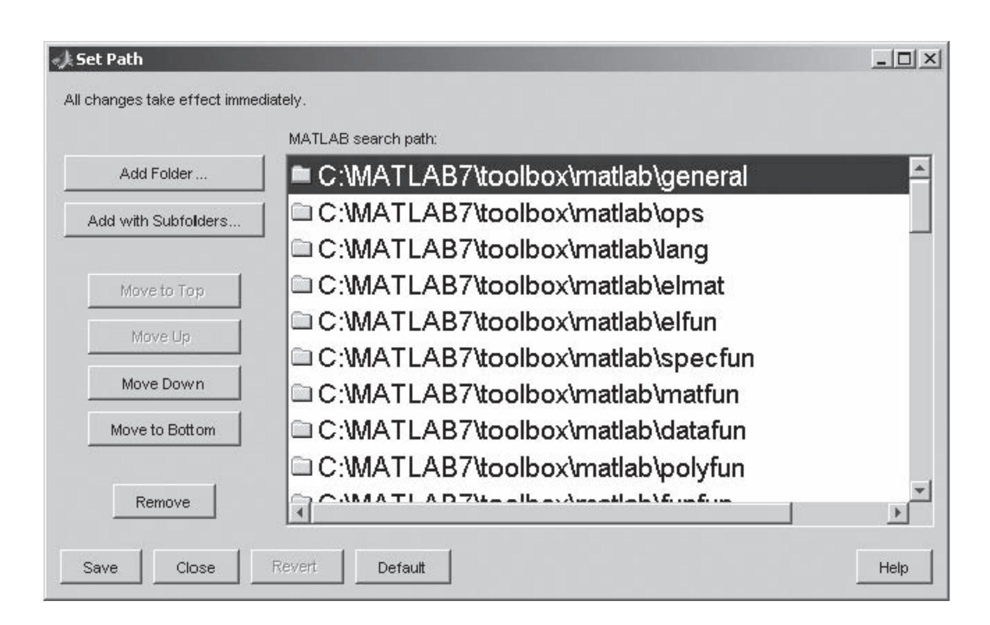


Figura 6.7
La herramienta path (ruta) le permite cambiar dónde MATLAB busca las definiciones de función.

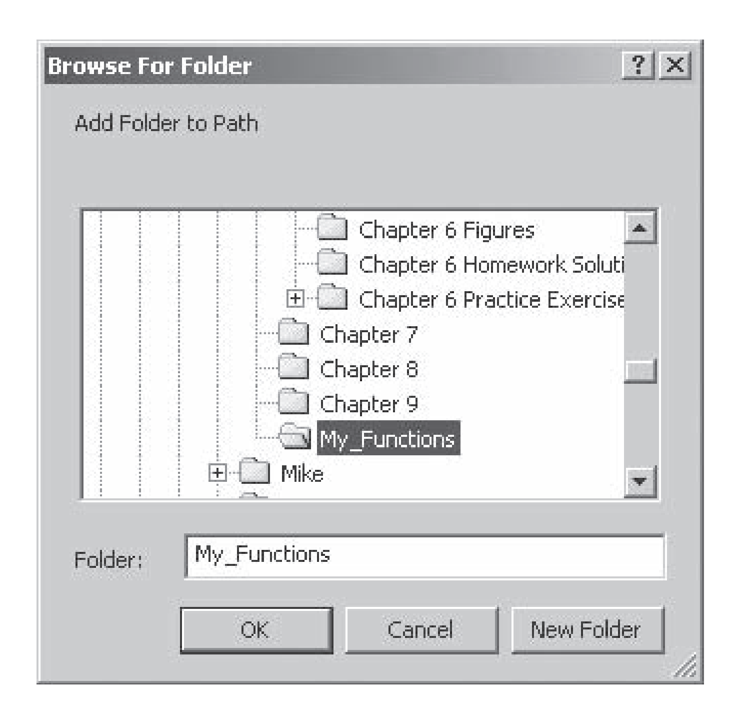


Figura 6.8
Ventana Browse for Folder (buscar carpeta).

debe hacer cambios permanentes a una computadora pública. Sin embargo, si alguien más hizo cambios que desea revertir, puede seleccionar el botón default, como se muestra en la figura 6.9, para regresar la ruta de búsqueda a su configuración original.

La herramienta path le permite cambiar interactivamente la ruta de búsqueda MATLAB; sin embargo, la función **addpath** le permite insertar la lógica para agregar una ruta de búsqueda a cualquier programa MATLAB. Consulte

```
help addpath
```

si desea modificar la ruta de esta forma.

MATLAB proporciona acceso a varias cajas de herramientas desarrolladas en The Math Works o por la comunidad de usuarios. Para más información, vea el website de la empresa, www.mathworks.com.

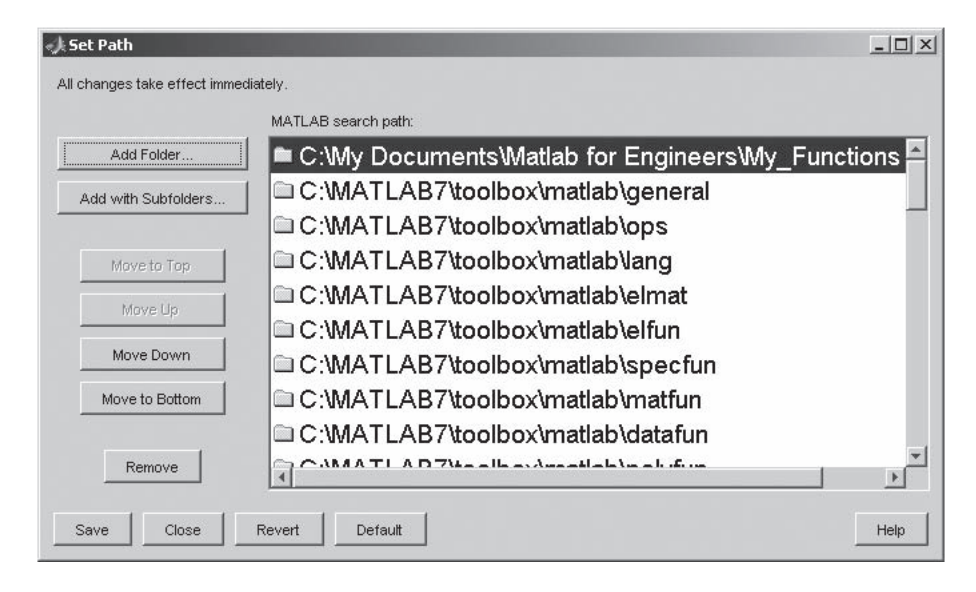


Figura 6.9
Ruta de búsqueda MATLAB modificada.

6.3 FUNCIONES ANÓNIMAS

Por lo general, si se enfrenta al problema de crear una función, querrá almacenarla para usar en otros proyectos de programación. Sin embargo, MATLAB incluye un tipo más simple de función, llamada *función anónima*. Nuevas en MATLAB 7, las funciones anónimas se definen en la ventana de comandos o en un archivo-m script y están disponibles, en gran medida como los nombres de variables, sólo hasta que se limpia el área de trabajo. Para crear una función anónima, considere el siguiente ejemplo:

Idea clave: las funciones anónimas se pueden incluir en los programas de archivo-m con otros comandos, o se pueden definir desde la ventana de comandos.

```
ln = @(x) log(x)
```

- El símbolo @ alerta a MATLAB que **ln** es una función.
- Inmediatamente después del símbolo @, se menciona la entrada a la función.
- Finalmente, se define la función.

El nombre de función aparece en la ventana de variables, mencionada como una function_handle (manipulador de función):

Nombre	Valor	Tamaño	Bytes	Clase
@ ln	@(x) log(x)	1x1	16	function_handle

Las funciones anónimas se pueden usar como cualquier otra función; por ejemplo,

```
ln(10)
ans =
      2.3026
```

Una vez que se limpia el área de trabajo, la función anónima ya no existe más. Las funciones anónimas y los manipuladores de función relacionados son útiles en funciones que requieren otras funciones como entrada (funciones de función).

Las funciones anónimas se pueden guardar como archivos .mat, tal como cualquier variable, y se pueden restaurar con el comando **load**.

MATLAB también soporta un tipo de función similar llamado función en línea. Al acceder al menú help se puede encontrar información acerca de las funciones en línea:

```
help inline
```

Las funciones en línea no ofrecen ventajas sobre las funciones anónimas y tienen una sintaxis ligeramente más complicada.

6.4 FUNCIONES DE FUNCIÓN

Idea clave: las funciones de función requieren que las funciones o función se manipulen como entrada.

Un ejemplo de una función de función interna MATLAB es la gráfica de función, **fplot**. Esta función requiere dos entradas: una función o un manipulador de función, y un rango sobre el cual graficar. Se puede demostrar el uso de **fplot** con el manipulador de función **ln**, que se define como

```
ln = @(x) log(x)
```

El manipulador de función se puede usar ahora como entrada a la función **fplot**:

```
fplot(ln,[0.1, 10])
```

El resultado se muestra en la figura 6.10.

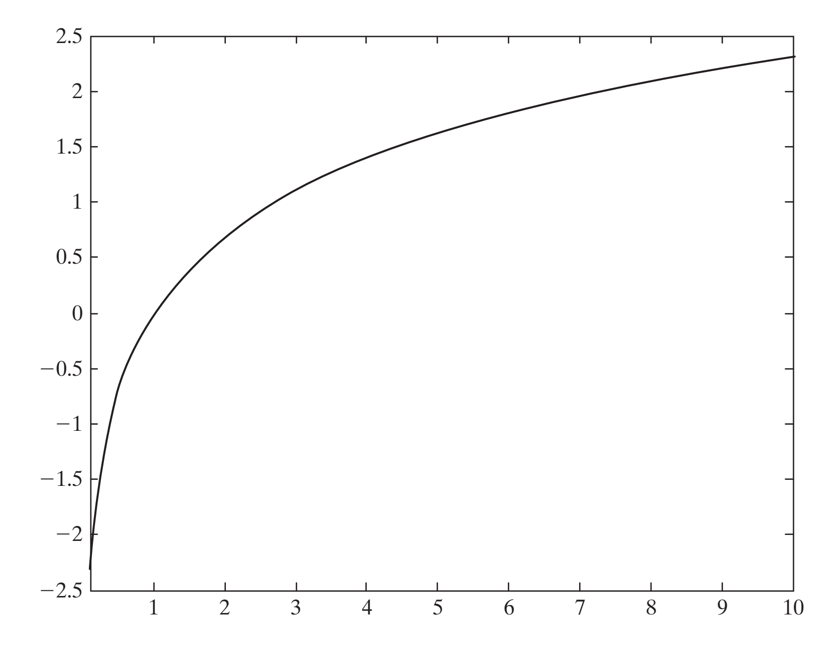


Figura 6.10
Los manipuladores de función se pueden usar como entrada a una función de función, como **fplot**.

RESUMEN

MATLAB contiene una gran variedad de funciones internas. Sin embargo, con frecuencia es útil crear las propias funciones MATLAB. El tipo más común de función MATLAB definida por el usuario es el archivo-m de función, que debe comenzar con una línea de definición de función que contenga

- la palabra **function**,
- una variable que defina la salida de función,
- un nombre de función, y
- una variable que se use para el argumento de entrada.

Por ejemplo,

```
function  output = my_function(x)
```

El nombre de función también debe ser el nombre del archivo-m en el que la función se almacena. Los nombres de función siguen las reglas de nomenclatura MATLAB estándar.

Al igual que las funciones internas, las funciones definidas por el usuario pueden aceptar entradas múltiples y pueden regresar resultados múltiples.

Se pueden acceder a los comentarios que siguen inmediatamente a la línea de definición de función desde la ventana de comandos con el comando **help**.

Las variables que se definen dentro de una función son locales a dicha función. No se almacenan en el área de trabajo y no se puede acceder a ellas desde la ventana de comandos. Las variables globales se pueden definir con el comando **global** que se usa tanto en la ventana de comandos (o archivo-m script) como en una función MATLAB. El buen estilo de programación sugiere que las variables globales se definan con letras mayúsculas. Sin embargo, en general, no es aconsejable usar variables globales.

Los grupos de funciones definidas por el usuario, llamados "cajas de herramientas", se pueden almacenar en un directorio común y acceder a ellas mediante la modificación de la ruta de búsqueda MATLAB. Esto se logra interactivamente con la herramienta path, o desde la barra de menú, como en

File → Set Path

o desde la línea de comando, con

```
pathtool
```

MATLAB proporciona acceso a numerosas cajas de herramientas desarrolladas en The MathWorks o por la comunidad de usuarios.

Otro tipo de función es la función anónima, que se define en una sesión MATLAB o en un archivo-m script y existe sólo durante dicha sesión. Las funciones anónimas son especialmente útiles para expresiones matemáticas muy simples o como entrada a las funciones de función más complicadas.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona y describe brevemente todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres especiales	
`@`	identifica un manipulador de función, como el que se usa con las funciones en línea
`%`	comentario

Comandos y funciones	
`addpath`	agrega un directorio a la ruta de búsqueda MATLAB
`function`	identifica un archivo-m como función
`meshgrid`	mapea dos vectores de entrada en dos matrices bidimensionales
`nargin`	determina el número de argumentos de entrada en una función
`nargout`	determina el número de argumentos de salida de una función
`pathtool`	abre la herramienta de ruta interactiva
`varargin`	indica que un número variable de argumentos puede ser entrada a una función

TÉRMINOS CLAVE

anónima
archivo-m
argumento
argumento de entrada
caja de herramientas
carpeta
comentarios
directorio
en línea
función
función de función
manipulador de función
nombre de archivo
nombre de función
variable global
variable local

PROBLEMAS

Archivos-m de función

Conforme cree funciones en esta sección, asegúrese de comentarlas de manera adecuada. Recuerde que, aunque muchos de estos problemas se podrían resolver sin una función, el objetivo de este capítulo es aprender a escribir y usar funciones.

6.1 Como se describió en el ejemplo 6.2, los metales en realidad son materiales cristalinos. Los cristales metálicos se llaman granos. Cuando el tamaño de grano promedio es pequeño, el metal es fuerte; cuando es grande, el metal es más débil. Dado que cada cristal en una muestra particular de metal es de un tamaño diferente, no es obvio cómo se podría describir el tamaño de cristal promedio. La American Society for Testing and Materials (ASTM) desarrolló la siguiente correlación para estandarizar las mediciones de tamaño de grano:

$$N = 2^{n-1}$$

El tamaño de grano ASTM (n) se determina al observar una muestra de metal bajo un microscopio a una amplificación de 100 $\times$ (potencia 100). Se estima (N) el número de granos en un área de 1 pulgada cuadrada (dimensiones reales de 0.01 pulgada $\times$ 0.01 pulgada) y se usa en la ecuación precedente para encontrar el tamaño de grano ASTM.

(a) Escriba una función MATLAB llamada **num_grains** para encontrar el número de granos en un área de 1 pulgada cuadrada (N) a una amplificación de 100$\times$ cuando el tamaño de grano ASTM se conoce.
(b) Use su función para encontrar el número de granos para tamaños de grano ASTM n = 10 a 100.
(c) Cree una gráfica de sus resultados.

6.2 Acaso la ecuación más famosa en física sea

$$E = mc^2$$

que relaciona la energía E con la masa m. La rapidez de la luz en el vacío, c, es la propiedad que vincula a las dos. La rapidez de la luz en el vacío es 2.9979×10^8 m/s.

(a) Cree una función llamada **energy** para encontrar la energía correspondiente a una masa dada en kg. Su resultado estará en joules, pues 1 kg m^2/s^2 = 1 joule.
(b) Use su función para encontrar la energía correspondiente a masas desde 1 kg hasta 10^6 kg. Use la función **logspace** (consulte **help/logspace**) para crear un vector masa adecuado.
(c) Cree una gráfica de sus resultados. Intente usar diferentes enfoques de graficación logarítmica (por ejemplo: **semilogy**, **semilogx** y **loglog**) para determinar la mejor forma de graficar sus resultados.

6.3 En química de primer año, se introduce la relación entre moles y masa

$$n = \frac{m}{\text{MW}}$$

donde

n = número de moles de una sustancia,
m = masa de la sustancia, y
MW = peso molecular (masa molar) de la sustancia.

(a) Cree un archivo-m de función llamado **nmoles** que requiera dos entradas vectoriales (la masa y el peso molecular) y que regrese el correspondiente número de moles. Puesto que proporciona entrada vectorial, será necesario usar la función **meshgrid** en sus cálculos.
(b) Ponga a prueba su función para los compuestos que se muestra en la tabla siguiente, para masas desde 1 hasta 10 g:

Compuesto	Peso molecular (masa molar)
Benceno	78.115 g/mol
Alcohol etílico	46.07 g/mol
Refrigerante R134a (tetrafluoroetano)	102.3 g/mol

Su resultado debe ser una matriz de 10 $\times$ 3.

6.4 Al reordenar la relación anterior entre moles y masa, puede encontrar la masa si conoce el número de moles de un compuesto:

$$m = n \times \text{MW}$$

(a) Cree un archivo-m de función llamado **mass** que requiere dos entradas vectoriales (el número de moles y el peso molecular) y que regrese la masa correspondiente. Puesto que proporciona entrada vectorial, será necesario usar la función **meshgrid** en sus cálculos.

(b) Ponga a prueba su función con los compuestos que se mencionan en el problema anterior, para valores de n desde 1 hasta 10.

6.5 La distancia hasta el horizonte aumenta conforme usted asciende una montaña (o una colina). La expresión

$$d = \sqrt{2rh + h^2}$$

donde

d = distancia hasta el horizonte,
r = radio de la Tierra, y
h = altura de la colina

se puede usar para calcular dicha distancia. La distancia depende de cuán alta sea la colina y del radio de la Tierra (u otro cuerpo planetario).

(a) Cree un archivo-m de función llamado **distance** para encontrar la distancia hasta el horizonte. Su función debe aceptar dos entradas vectoriales (radio y altura) y debe regresar la distancia hasta el horizonte. No olvide que necesitará usar **meshgrid** porque sus entradas son vectores.

(b) Cree un programa MATLAB que use su función distance para encontrar la distancia en millas hasta el horizonte, tanto en la Tierra como en Marte, para colinas desde 0 hasta 10,000 pies. Recuerde usar unidades consistentes en sus cálculos. Note que

- Diámetro de la Tierra = 7926 millas.
- Diámetro de Marte = 4217 millas.

Reporte sus resultados en una tabla. Cada columna debe representar un planeta diferente y cada fila debe representar una altura de colina diferente.

6.6 Un cohete se lanza verticalmente. En el tiempo $t = 0$, el motor del cohete se apaga. En ese momento, el cohete ha alcanzado una altura de 500 metros y se eleva con una velocidad de 125 metros por segundo. Entonces la gravedad toma el control. La altura del cohete como función del tiempo es

$$h(t) = -\frac{9.8}{2}t^2 + 125t + 500 \quad \text{para} \quad t > 0$$

(a) Cree una función llamada **height** que acepte tiempo como entrada y regresa la altura del cohete. Use su función en sus soluciones a las partes b y c.

(b) Grafique **height** contra tiempo para tiempos desde 0 hasta 30 segundos. Use un incremento de 0.5 segundo en su vector tiempo.

(c) Encuentre el tiempo cuando el cohete comienza a caer de vuelta al suelo. (En este ejercicio será útil la función **max**.)

6.7 La distancia que recorre un cuerpo en caída libre es

$$x = \frac{1}{2}gt^2$$

donde

g = aceleración debida a la gravedad, 9.8 m/s^2,
t = tiempo en segundos,
x = distancia recorrida en metros.

Si ya cursó cálculo, sabe que se puede encontrar la velocidad del objeto al tomar la derivada de la ecuación anterior. Esto es,

$$\frac{dx}{dt} = v = gt$$

Se puede encontrar la aceleración al tomar la derivada de nuevo:

$$\frac{dv}{dt} = a = g$$

(a) Cree una función llamada **free_fall** con un solo vector de entrada **t** que regrese valores para distancia **x**, velocidad **v** y aceleración **g**.
(b) Ponga a prueba su función con un vector tiempo que varíe desde 0 hasta 20 segundos.

6.8 Cree una función llamada **polygon** que dibuje un polígono con cualquier número de lados. Su función debe requerir una sola entrada: el número de lados deseado. No debe regresar valor alguno a la ventana de comandos, sino que debe dibujar el polígono solicitado en coordenadas polares.

Creación de su propia caja de herramientas

6.9 Este problema requiere que usted genere tablas de conversión de temperatura. Use las siguientes ecuaciones, que describen las relaciones entre temperaturas en grados Fahrenheit (T_F), grados Celsius (T_C), grados Kelvin (T_K) y grados Rankine (T_R), respectivamente:

$$T_F = T_R - 459.67\ °\text{R}$$

$$T_F = \frac{9}{5}T_C + 32\ °\text{F}$$

$$T_R = \frac{9}{5}T_K$$

Necesitará reordenar estas expresiones para resolver algunos de los problemas.

(a) Cree una función llamada **F_to_K** que convierta temperaturas en Fahrenheit a Kelvin. Use su función para generar una tabla de conversión para valores desde 0 °F hasta 200 °F.
(b) Cree una función llamada **C_to_R** que convierta temperaturas en Celsius a Rankine. Use su función para generar una tabla de conversión desde 0 °C hasta 100 °C. Imprima 25 líneas en la tabla. (Use la función **linspace** para crear su vector de entrada.)
(c) Cree una función llamada **C_to_F** que convierta temperaturas en Celsius a Fahrenheit. Use su función para generar una tabla de conversión desde 0 °C hasta 100 °C. Elija un espaciamiento adecuado.
(d) Agrupe sus funciones en una carpeta (directorio) llamado **my_temp_conversions**. Ajuste la ruta de búsqueda MATLAB de modo que encuentre su carpeta. (¡No guarde cambios en una computadora pública!)

Funciones anónimas

6.10 Los barómetros se han usado durante casi 400 años para medir cambios de presión en la atmósfera. El primer barómetro conocido lo inventó Evangelista Torricelli (1608-1647), quien fue estudiante de Galileo en Florencia, Italia, durante sus años finales. La altura de un líquido en un barómetro es directamente proporcional a la presión atmosférica, o

$$P = \rho g h$$

donde P es la presión, ρ es la densidad del fluido del barómetro y h es la altura de la columna de líquido. Para barómetros de mercurio, la densidad del fluido es 13,560 kg/m^3. En la superficie de la Tierra, la aceleración debida a la gravedad, g, es 9.8 m/s^2. Por tanto, la única variable en la ecuación es la altura de la columna de fluido, h, que debe tener la unidad de metros.

(a) Cree una función anónima **P** que encuentre la presión si se proporciona el valor de h. Las unidades de su respuesta serán

$$\frac{\text{kg}}{\text{m}^3}\frac{\text{m}}{\text{s}^2}\text{m} = \frac{\text{kg}}{\text{m}}\frac{1}{\text{s}^2} = \text{Pa}$$

(b) Cree otra función anónima para convertir presión en Pa (pascales) a presión en atmósferas (atm). Llame a la función **Pa_to_atm**. Note que

$$1 \text{ atm} = 101{,}325 \text{ Pa}$$

(c) Use sus funciones anónimas para encontrar la presión para alturas de fluido desde 0.5 m hasta 1.0 m de mercurio.
(d) Guarde sus funciones anónimas como archivos **.mat**.

6.11 La energía requerida para calentar agua a presión constante es aproximadamente igual a

$$E = mC_p\Delta T$$

donde
m = masa del agua en gramos,
C_p = capacidad calorífica del agua, 1 cal/g °K, y
ΔT = cambio en temperatura, °K.

(a) Cree una función anónima llamada **heat** para encontrar la energía requerida para calentar 1 gramo de agua si el cambio en temperatura se proporciona como entrada.
(b) Su resultado estará en calorías:

$$\text{g}\frac{\text{cal}}{\text{g}}\frac{1}{\text{K}}\text{K} = \text{cal}$$

Los joules son la unidad de energía usada con más frecuencia en ingeniería. Cree otra función anónima **cal_to_J** para convertir su respuesta de la parte (a) en joules. (Existen 4.2 joules/cal.)
(c) Guarde sus funciones anónimas como archivos **.mat**.



CAPÍTULO 7

Entrada y salida controladas por el usuario

Objetivos

Después de leer este capítulo, el alumno será capaz de

- indicar al usuario que ingrese entrada a un programa de archivo-m.
- crear salida con la función **disp**.
- crear salida formateada con **fprintf**.
- usar técnicas gráficas para proporcionar entrada al programa.
- usar el modo cell para modificar y correr programas de archivo-m.

INTRODUCCIÓN

Hasta el momento, MATLAB se ha usado de dos maneras: como una memoria de trabajo auxiliar (scratch pad) en la ventana de comandos y para escribir programas simples (archivos-m script y funciones) en la ventana de edición. En ambos casos se supuso que el programador era el usuario. En este capítulo se avanzará hacia los programas más complicados, escritos en la ventana de edición, suponiendo que el programador y el usuario pueden ser personas diferentes. Esto hará necesario comunicarse con el usuario a través de comandos de entrada y salida, en lugar de reescribir el código real para resolver problemas similares. MATLAB ofrece funciones internas para permitir al usuario comunicarse con un programa conforme se ejecuta. El comando **input** detiene el programa y conmina al usuario a ingresar una entrada; los comandos **disp** y **fprintf** proporcionan salida a la ventana de comandos.

7.1 ENTRADA DEFINIDA POR EL USUARIO

Aunque se han escrito programas en archivos-m script, se supuso que el programador (usted) y el usuario eran la misma persona. Para correr el programa con diferentes valores de entrada, en realidad se cambió parte del código. Se pueden recrear programas más generales al permitir al usuario ingresar valores de una matriz desde el teclado mientras el programa corre. La función **input** le permite hacer esto. Despliega una cadena de texto en la ventana de comando y luego espera que el usuario proporcione la entrada solicitada. Por ejemplo,

```
z = input('Ingrese un valor')
```

despliega

```
Ingrese un valor
```

en la ventana de comandos. Si el usuario ingresa un valor como

```
5
```

el programa asigna el valor 5 a la variable **z**. Si el comando **input** no termina con un punto y coma, el valor ingresado se despliega en la pantalla:

```
z =
       5
```

El mismo enfoque se puede usar para ingresar una matriz uni o bidimensional. El usuario debe proporcionar los paréntesis y delimitadores adecuados (comas y puntos y coma). Por ejemplo,

```
z = input('Ingrese valores para z entre corchetes')
```

Idea clave: la función input se puede usar para comunicarse con el usuario del programa.

pide al usuario que ingrese una matriz como

```
[ 1, 2, 3; 4, 5, 6]
```

y responde con

```
z =
     1  2  3
     4  5  6
```

Este valor de entrada de **z** se puede usar entonces en cálculos subsecuentes por el archivo-m script.

Los datos ingresados con **input** no necesitan ser información numérica. Suponga que se conmina al usuario con el comando

```
x=input('Ingrese su nombre en apóstrofes')
```

e ingrese

```
'Holly'
```

cuando se le indique. Puesto que no se usó punto y coma al final del comando **input**, MATLAB responderá

```
x =
  Holly
```

Note en la ventana del área de trabajo que **x** se menciona como un arreglo carácter de 1×5:

Nombre	Valor	Tamaño	Bytes	Clase
abc x	**'Holly'**	**1 × 5**	**6**	**char**

Si ingresa una cadena (en MATLAB, las cadenas son arreglos carácter), debe encerrar los caracteres en apóstrofes. Sin embargo, una forma alternativa del comando input alerta a la función a esperar entrada carácter sin los apóstrofes, al especificar entrada cadena en el segundo campo:

```
x=input('Ingrese su nombre', 's')
```

Ahora sólo necesita ingresar los caracteres, como

```
Ralph
```

y el programa responde con

```
x =
  Ralph
```

Ejercicio de práctica 7.1

1. Cree un archivo-m para calcular el área A de un triángulo:

$$A = \frac{1}{2}\text{ base altura}$$

   Conmine al usuario a ingresar los valores para la base y la altura.
2. Cree un archivo-m para encontrar el volumen V de un cilindro circular recto:

$$V = \pi r^2 h$$

   Conmine al usuario a ingresar los valores de r y h.
3. Cree un vector desde 1 hasta n, y permita al usuario ingresar el valor de n.
4. Cree un vector que comience en a, termine en b y tenga un espaciamiento de c. Permita al usuario ingresar todos estos parámetros.

EJEMPLO 7.1

Objetos en caída libre

Analice el comportamiento de un objeto en caída libre. (Véase la figura 7.1.) La ecuación relevante es

$$d = \frac{1}{2}gt^2$$

donde d = distancia que recorre el objeto,
g = aceleración debida a la gravedad, y
t = tiempo en que el objeto recorre la distancia d.



Figura 7.1
Torre inclinada de Pisa. (Cortesía de Tim Galligan.)

Se debe permitir al usuario especificar el valor de g, la aceleración debida a la gravedad, y un vector de valores de tiempo.

1. Establezca el problema.
   Encontrar la distancia que recorre un objeto en caída libre y graficar los resultados.
2. Describir las entradas y salidas.

   Entrada Valor de g, la aceleración debida a la gravedad, proporcionada por el usuario
   Tiempo, proporcionado por el usuario

   Salida Distancias
   Gráfica de distancia contra tiempo

3. Desarrolle un ejemplo a mano.

$$d = \frac{1}{2}gt^2\text{, de modo que, en la luna, a 100 segundos,}$$

$$d = \frac{1}{2} \times 1.6\text{ m/s}^2 \times 100^2\text{ s}^2$$

$$d = 8000\text{ m}$$

4. Desarrolle una solución MATLAB.

```
%Example 7.1
%Free fall
```

```
clear, clc
%Solicite entrada del usuario
g = input('¿Cuál es el valor de aceleración debida
a gravedad?')
start=input('¿Qué tiempo de inicio le gustaría? ')
finish = input('¿Qué tiempo final le gustaría? ')
incr = input('¿Qué incrementos de tiempo le gustaría
  calcular? ')
t=start:incr:finish;
%Calcula la distancia
d=1/2*g*t.^2;
%Grafica los resultados
loglog(t,d)
title('Distancia recorrida en caída libre')
xlabel('tiempo, s'),ylabel('distancia, m')
%Encuentra la distancia máxima recorrida
final_distance = max(d)
```

La interacción en la ventana de comandos es la siguiente:

```
¿Cuál es el valor de aceleración debida a gravedad? 1.6
g =
  1.6000
¿Qué tiempo de inicio le gustaría? 0
start =
  0
¿Qué tiempo final le gustaría? 100
finish =
  100
¿Qué incrementos de tiempo le gustaría calcular? 10
incr =
  10
final_distance =
  8000
```

En la figura 7.2 se grafican los resultados del ejemplo 7.1.

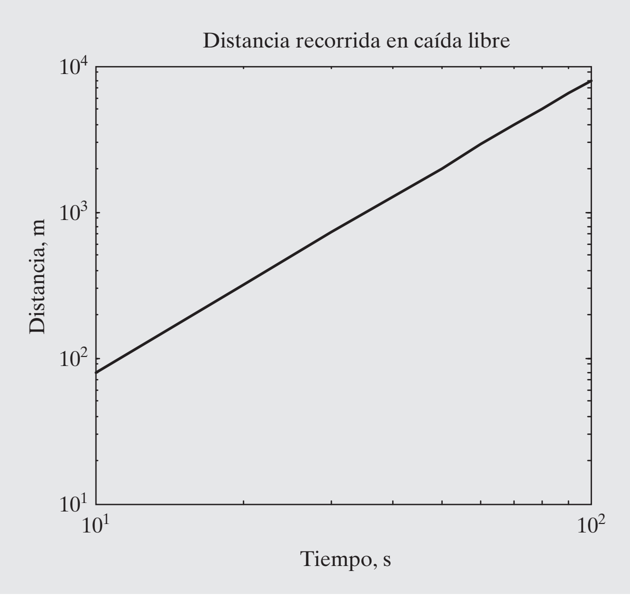


Figura 7.2
Distancia recorrida cuando la aceleración es 1.6 m/s².

5. Ponga a prueba la solución.
   Compare la solución MATLAB con la solución a mano. Dado que el usuario puede controlar la entrada, se ingresan los datos usados en la solución a mano. MATLAB dice que la distancia final recorrida es 8000 m, la cual corresponde a la distancia recorrida luego de 100 segundos, puesto que se ingresó 100 segundos como el tiempo final.

7.2 OPCIONES DE SALIDA

Existen muchas formas de desplegar los contenidos de una matriz. La más simple es ingresar el nombre de la matriz, sin punto y coma. El nombre de la matriz se repetirá y los valores de la matriz se desplegarán comenzando en la línea siguiente. Por ejemplo, defina primero una matriz **x**:

```
x = 1:5;
```

Puesto que hay punto y coma al final del enunciado de asignación, los valores en **x** no se repiten en la ventana de comandos. Sin embargo, si quiere desplegar **x** más tarde en su programa, simplemente escriba en el nombre de variable

```
x
```

lo que regresa

```
x =
      1     2     3     4     5
```

MATLAB ofrece otros dos enfoques para desplegar resultados: la función **disp** y la función **fprintf**.

Idea clave: la función disp puede desplegar arreglos carácter o arreglos numéricos.

7.2.1 Función despliegue (display)

La función despliegue (display, **disp**) se puede usar para desplegar los contenidos de una matriz sin imprimir el nombre de matriz. Por lo tanto,

```
disp(x)
```

regresa

```
1     2     3     4     5
```

El comando display también se puede usar para desplegar una cadena (texto encerrado en marcas de comilla simple o apóstrofe). Por ejemplo,

```
disp('Los valores en la matriz x son:');
```

regresa

```
Los valores en la matriz x son:
```

Cuando ingresa una cadena como entrada en la función **disp**, en realidad ingresa un arreglo de información carácter. Intente ingresar lo siguiente en la línea de comando:

```
'Los valores en la matriz x son:'
```

MATLAB responde

```
ans =
'Los valores en la matriz x son:'
```

arreglo carácter: almacena información carácter

La ventana del área de trabajo cita **ans** como un arreglo carácter 1 × 32.

Nombre	Tamaño	Bytes	Clase
abc ans	1 × 32	90	char array

Los arreglos carácter almacenan información carácter en arreglos similar a los arreglos numéricos. Los caracteres pueden ser letras, números, puntuación e incluso algunos caracteres no desplegables. Cada carácter, incluidos espacios, es un elemento en el arreglo carácter.

Cuando se ejecutan las dos funciones de despliegue

```
disp('Los valores en la matriz x son:');
disp(x)
```

Idea clave: los caracteres pueden ser letras, números o símbolos.

MATLAB responde

```
Los valores en la matriz x son:
1   2   3   4   5
```

Note que las dos funciones **disp** se despliegan en líneas separadas. Puede evitar esta característica al crear una matriz combinada de sus dos salidas, con el uso de la función **num2str** (número a cadena). El proceso se llama concatenación y crea un solo arreglo carácter. Por ende,

```
disp(['Los valores en el arreglo x son: ' num2str(x)])
```

regresa

```
Los valores en el arreglo x son: 1 2 3 4 5
```

La función **num2str** cambia un arreglo de números en un arreglo de caracteres. En el ejemplo anterior se usó **num2str** para transformar la matriz **x** a un arreglo carácter, que entonces se combinó con la primera cadena (mediante corchetes []) para hacer un arreglo carácter más grande. Puede ver la matriz resultante al escribir

```
A = ['Los valores en la matriz x son: ' num2str(x)]
```

que regresa

```
A =
       Los valores en el arreglo x son: 1 2 3 4 5
```

Al verificar en la ventana del área de trabajo, se ve que **A** es una matriz 1 × 45. La ventana del área de trabajo también indica que la matriz contiene datos carácter en lugar de información numérica. Esto se evidencia tanto por el icono en frente de **A** como en la columna class.

Nombre	Tamaño	Bytes	Clase
abc A	1 × 45	90	char array

Sugerencia

Si quiere incluir un apóstrofe en una cadena, necesita ingresar el apóstrofe dos veces. Si no lo hace, MATLAB interpretará el apóstrofe como terminación de la cadena. Un ejemplo del uso de dos apóstrofes es

```
disp('The moon''s gravity is 1/6th that of the earth')
```

Puede usar una combinación de funciones **input** y **disp** para imitar una conversación. Intente crear y correr el siguiente archivo-m:

```
disp('Hi There');
disp('I''m your MATLAB program');
name=input('Who are you?','s');
disp(['Hi ',name]);
answer=input('Don''t you just love computers?', 's');
disp([answer,'?']);
disp('Computers are very useful');
disp('You''ll use them a lot in college!!');
disp('Good luck with your studies')
pause(2);
disp('Bye bye')
```

7.2.2 Salida formateada

La función **fprintf** (función impresión formateada) le da incluso mayor control sobre la salida de la que tiene con la función **disp**. Además de desplegar valores tanto texto como matriz, puede especificar el formato a usar al desplegar los valores, y puede especificar cuándo saltar a una nueva línea. Si usted es programador C, estará familiarizado con la sintaxis de esta función. Con pocas excepciones, la función MATLAB **fprintf** usa las mismas especificaciones de formateo que la función **fprintf** C. Esto difícilmente es sorprendente, pues MATLAB se escribió en C. (MATLAB originalmente se escribió en FORTRAN y más tarde se rescribió en C.)

La forma general del comando **fprintf** contiene dos argumentos, uno cadena y otro una lista de matrices:

```
fprintf(format-string, var,...)
```

Considere el siguiente ejemplo:

```
cows = 5;
fprintf(''Hay %f vacas en el pastizal', vacas)
```

La cadena, que es el primer argumento dentro de la función **fprintf**, contiene un marcador de posición (placeholder) (%) donde se insertará el valor de la variable (en este caso, **vacas**). El marcador de posición también contiene información de formateo. En este ejemplo, **%f** le pide a MATLAB desplegar el valor de **vacas** en un formato de punto fijo por defecto. El formato por defecto despliega seis lugares después del punto decimal:

```
Hay 5.000000 vacas en el pastizal
```

Además del formato de punto fijo por defecto, MATLAB le permite especificar un formato exponencial, **%e**, o le permite a MATLAB elegir el que sea

Tabla 7.1 Formato de tipo de campo

Tipo de campo	Resultado
%f	notación punto fijo o decimal
%e	notación exponencial
%g	la que sea más corta, **%f** o **%e**
%c	información carácter
%s	cadena de caracteres

En la característica help se describen tipos de campos adicionales.

más corto, punto fijo o exponencial (**%g**). También le permite desplegar información carácter (**%c**) o una cadena de caracteres (**%s**). La tabla 7.1 ilustra varios formatos.

Idea clave: la función fprintf le permite controlar cómo se despliegan los números.

MATLAB no comienza automáticamente una nueva línea después de ejecutar una función **fprintf**. Si usted intentó el ejemplo de comando **fprintf** anterior, probablemente notó que el prompt del comando está en la misma línea que la salida:

```
Hay 5.000000 vacas en el pastizal >>
```

Si ejecuta otro comando, los resultados aparecerán en la misma línea en lugar de moverlos abajo. Por tanto, si se emiten los nuevos comandos

```
cow = 6;
fprintf(''Hay %f vacas en el pastizal', vacas);
```

desde un archivo-m, MATLAB continúa el despliegue de ventana de comandos en la misma línea:

```
Hay 5.000000 vacas en el pastizal Hay 6.000000
vacas en el pastizal
```

Idea clave: la función fprintf le permite desplegar tanto caracteres como información numérica con un solo comando.

Para hacer que MATLAB inicie una nueva línea, necesitará usar **\n**, llamado salto de línea (linefeed), al final de la cadena. Por ejemplo, el código

```
cows=5;
fprintf('Hay %f vacas en el pastizal \n', vacas)
cows = 6;
fprintf('Hay %f vacas en el pastizal \n', vacas)
```

regresa la siguiente salida:

```
Hay 5.000000 vacas en el pastizal
Hay 6.000000 vacas en el pastizal
```

Sugerencia

La diagonal inversa (\) y la diagonal normal (/) son caracteres diferentes. Es un error común confundirlas, ¡y luego el comando linefeed no funciona! En vez de ello, la salida en la ventana de comandos será

```
Hay 5.000000 vacas en el pastizal /n
```

En la tabla 7.2 se mencionan otros comandos de formato especiales. El tabulador, tab (**\t**) es especialmente útil para crear tablas en las que todas las líneas ajusten limpiamente.

Tabla 7.2 Comandos de formato especial

Comando de formato	Acción resultante
`\n`	salto de línea (linefeed)
`\r`	regreso de carro (similar a linefeed)
`\t`	tabulador
`\b`	retroceder un espacio (backspace)

Puede controlar aún más cómo se despliegan las variables al usar los optativos **width field** y **precision field** con el comando format. El **width field** controla el número mínimo de caracteres a imprimir. Debe ser un entero decimal positivo. El **precision field** está precedido por un punto (**.**) y especifica el número de lugares decimales después del punto decimal para tipos exponencial y punto fijo. Por ejemplo, **%8.2f** especifica que el ancho total mínimo disponible para desplegar su resultado es ocho dígitos, dos de los cuales están después del punto decimal. Por tanto, el código

```
voltage = 3.5;
fprintf('El voltaje es %8.2f milivolts \n',voltaje);
```

regresa

```
El voltaje es     3.50 milivolts
```

Note el espacio vacío antes del número 3.50. Esto ocurre porque se reservaron seis espacios (ocho en total, dos después del decimal) para la porción del número a la izquierda del punto decimal.

Muchas veces, cuando usa la función **fprintf**, su variable será una matriz, por ejemplo,

```
x = 1:5;
```

MATLAB repetirá la cadena el comando **fprintf** hasta que use todos los valores en la matriz. Por tanto,

```
fprintf('%8.2f \n',x);
```

regresa

```
1.00
2.00
3.00
4.00
5.00
```

Si la variable es una matriz bidimensional, MATLAB usa los valores una columna a la vez, comenzando por la primera columna, luego la segunda columna, etcétera. He aquí un ejemplo más complicado:

```
feet = 1:3;
inches = feet.*12;
```

Combine estas dos matrices:

```
table = [feet;inches]
```

Entonces MATLAB regresa

```
table =
       1 2 3
       12 24 36
```

Ahora se puede usar la función **fprintf** para crear una tabla que sea más fácil de interpretar. Por ejemplo,

```
fprintf('%4.0f %7.2f \n',table)
```

envía la siguiente salida a la ventana de comandos:

```
1  12.00
2  24.00
3  36.00
```

La función **fprintf** puede aceptar un número variable de matrices después de la cadena. Usa todos los valores en cada una de dichas matrices, en orden, antes de moverse hacia la siguiente matriz. Como ejemplo, suponga que se quiere usar las matrices feet e inches sin combinarlas en la matriz tabla. Entonces se podría escribir

```
fprintf('%4.0f %7.2f \n', feet, inches)
  1   2.00
  3   12.00
  24  36.00
```

La función opera a través de los valores de **feet** primero y luego usa los valores en **inches**. Es improbable que esto sea lo que usted realmente quería que hiciera la función (no lo era en este ejemplo), de modo que los valores de salida casi siempre se agrupan en una sola matriz para usar en **fprintf**.

El comando **fprintf** le da considerablemente más control sobre la forma de su salida que los comandos de formato simple de MATLAB. Sin embargo, requiere cierto cuidado y prudencia usarlo.

Sugerencia

Uno de los errores más comunes cometido por los nuevos programadores es olvidar incluir la **f** en la secuencia del marcador de posición. La función **fprintf** no funciona, pero tampoco se envía mensaje de error.

Sugerencia

Si quiere incluir un signo de porcentaje en un enunciado **fprintf**, necesita ingresar % dos veces. Si no lo hace, MATLAB interpretará % como un marcador de posición para datos. Por ejemplo,

```
fprintf('La tasa de interés es %5.2f %% \n', 5)
```

resulta en

```
La tasa de interés es 5.00 %
```

EJEMPLO 7.2

Caída libre: salida formateada

Vuelva a hacer el ejemplo 7.1, pero esta vez cree una tabla de resultados en lugar de una gráfica, y use los comandos **disp** y **fprintf** para controlar la apariencia de la salida.

1. Establezca el problema.
   Encontrar la distancia recorrida por un objeto en caída libre.
2. Describa las entradas y salidas.

   Entrada Valor de g, la aceleración debida a la gravedad, proporcionada por el usuario
   Tiempo, t

   Salida Distancias calculadas para cada planeta y la Luna
3. Desarrolle un ejemplo a mano.

$$d = \frac{1}{2} gt^2, \text{ de modo que, en la Luna, a 100 segundos,}$$

$$d = \frac{1}{2} \times 1.6 \text{ m/s}^2 \times 100^2 \text{ s}^2$$

$$d = 8000 \text{ m}$$

4. Desarrolle una solución MATLAB.

```
%Ejemplo 7.2
%Caída libre
clear, clc
%Solicite entrada del usuario
g = input('¿Cuál es el valor de aceleración debida
  a gravedad?')
start=input('¿Qué tiempo de inicio le gustaría?')
finish = input('¿Qué tiempo final le gustaría?')
incr = input('¿Qué incrementos de tiempo le gustaría
  calcular?')
t=start:incr:finish;
%Calcula la distancia
d=1/2*g*t.^2;
%Crea una matriz de los datos de salida
table=[t;d];
%Envía la salida a la ventana de comandos
fprintf('Para una aceleración debida a la gravedad de %5.1f
segundos \n se calcularon los siguientes datos \n', g)
disp('Distancia recorrida en caída libre')
disp('tiempo, s distancia, m')
fprintf('%8.0f %10.2f\n',table)
```

Este archivo-m produce la siguiente interacción en la ventana de comandos:

```
¿Cuál es el valor de aceleración debida a gravedad?1.6
g =
  1.6000
```

```
¿Qué tiempo de inicio le gustaría?0
start =
  0

¿Qué tiempo final le gustaría?100
finish =
  100

¿Qué incrementos de tiempo le gustaría calcular?10
incr =
  10

Para una aceleración debida a la gravedad de 1.6 segundos
 se calcularon los siguientes datos

Distance Traveled in Free Fall
tiempo, s   distancia, m

     0          0.00
    10         80.00
    20        320.00
    30        720.00
    40       1280.00
    50       2000.00
    60       2880.00
    70       3920.00
    80       5120.00
    90       6480.00
   100       8000.00
```

5. Ponga a prueba la solución.
   Compare la solución MATLAB con la solución a mano. Dado que la salida es una tabla, es fácil ver que la distancia recorrida a 100 segundos es 8000 m. Intente usar otros datos como entrada y compare sus resultados con la gráfica producida en el ejemplo 7.1.

Ejercicio de práctica 7.2

En un archivo-m

1. Use el comando **disp** para crear un título para una tabla que convierte pulgadas a pies.
2. Use el comando **disp** para crear encabezados de columna para su tabla.
3. Cree un vector **inches** desde 0 hasta 120 con un incremento de 10.
   Calcule los correspondientes valores de **feet**.
   Agrupe el vector **inch** y el vector **feet** en una matriz **table**.
   Use el comando **fprintf** para enviar su tabla a la ventana de comandos.

7.3 ENTRADA GRÁFICA

MATLAB ofrece una técnica para ingresar gráficamente pares ordenados de valores x y y. El comando **ginput** permite al usuario seleccionar puntos desde una ventana de figura y convierte los puntos en las coordenadas apropiadas x y y. En el enunciado

```
[x,y] = ginput(n)
```

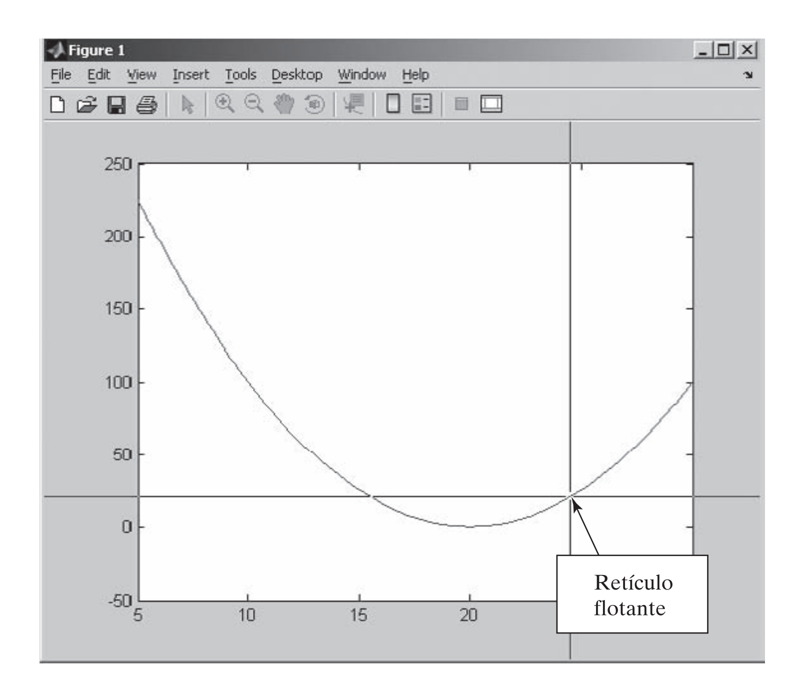


Figura 7.3
La función **ginput** permite al usuario escoger puntos de una gráfica.

MATLAB pide al usuario seleccionar *n* puntos de la ventana de figura. Si el valor de **n** no se incluye, como en

```
[x,y] = ginput
```

entonces MATLAB acepta puntos hasta que se ingresa la tecla return.

Esta técnica es útil para escoger puntos de una gráfica. Considere la gráfica en la figura 7.3. La figura se creó al definir *x* desde 5 hasta 30 y calcular *y*:

```
x = 5:30;
y = x.^2 - 40.*x + 400;
plot(x,y)
axis([5,30,-50,250])
```

Los valores de eje se definieron de modo que sería más fácil trazarlos.

Una vez que se ejecuta la función **ginput**, como en

```
[a,b] = ginput
```

MATLAB agrega un retículo flotante a la gráfica, como se muestra en la figura 7.3. Luego que este retículo se posiciona a satisfacción del usuario, seleccionar return (enter) envía los valores de las coordenadas *x* y *y* al programa:

```
a =
  24.4412
b =
  19.7368
```

7.4 USO DEL MODO CELDA EN ARCHIVOS-M DE MATLAB

MATLAB 7 tiene una utilidad nueva que permite al usuario dividir los archivos-m en secciones, o celdas (cell), que se pueden ejecutar una a la vez. Esta característica es particularmente

útil conforme usted desarrolla programas MATLAB. El modo cell también permite al usuario crear reportes en varios formatos que muestran los resultados del programa.

Idea clave: el modo cell (celda) es nuevo a MATLAB 7.

Para activar el modo cell, seleccione

Cell → Enable Cell Mode

en la barra de menú de la ventana de edición, como se muestra en la figura 7.4. Una vez habilitado el modo cell, aparece la barra de herramientas cell, como se muestra en la figura 7.5.

Para dividir su programa archivo-m en celdas, puede crear divisores de celda con un doble signo de porcentaje seguido por un espacio. Si quiere nombrar la celda, sólo agregue un nombre en la misma línea del divisor de celda:

```
%% Cell Name
```

Idea clave: el modo cell (celda) le permite ejecutar porciones del código incrementalmente.

Una vez que los divisores de celda están en su lugar, si mueve el cursor a alguna parte adentro de la celda, toda la celda se torna amarillo pálido. Por ejemplo, en la figura 7.5, las primeras tres líneas del programa archivo-m constituyen la primera celda. Ahora se pueden usar los iconos de evaluación en la barra de herramientas de celda para evaluar una sola sección, evaluar la sección actual y moverse a la siguiente sección, o evaluar toda la fila. También en la barra de herramientas de celda hay un icono que menciona todos los títulos de celda en el archivo-m, como se muestra en la figura 7.6. La tabla 7.3 muestra los iconos disponibles en la barra de herramientas de celda, junto con sus funciones.

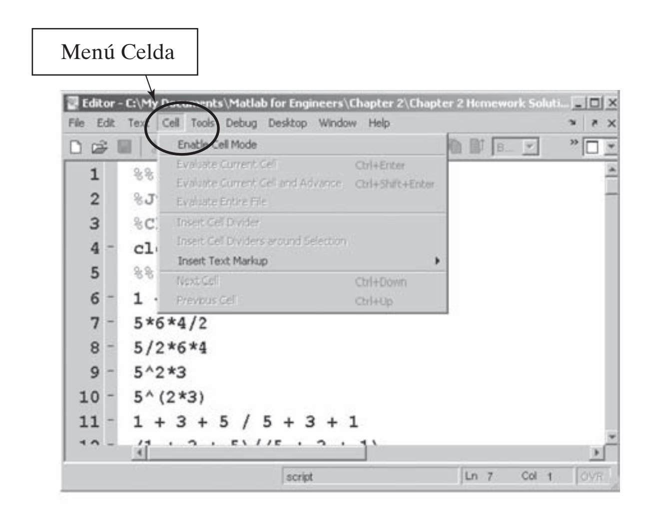


Figura 7.4
Puede acceder al modo celda desde la barra de menú en la ventana de edición.

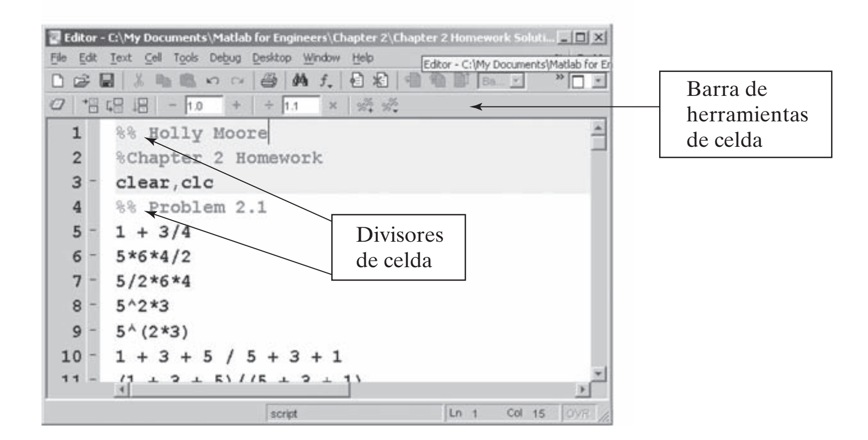


Figura 7.5
La barra de herramientas de celda permite al usuario ejecutar una celda o sección a la vez.

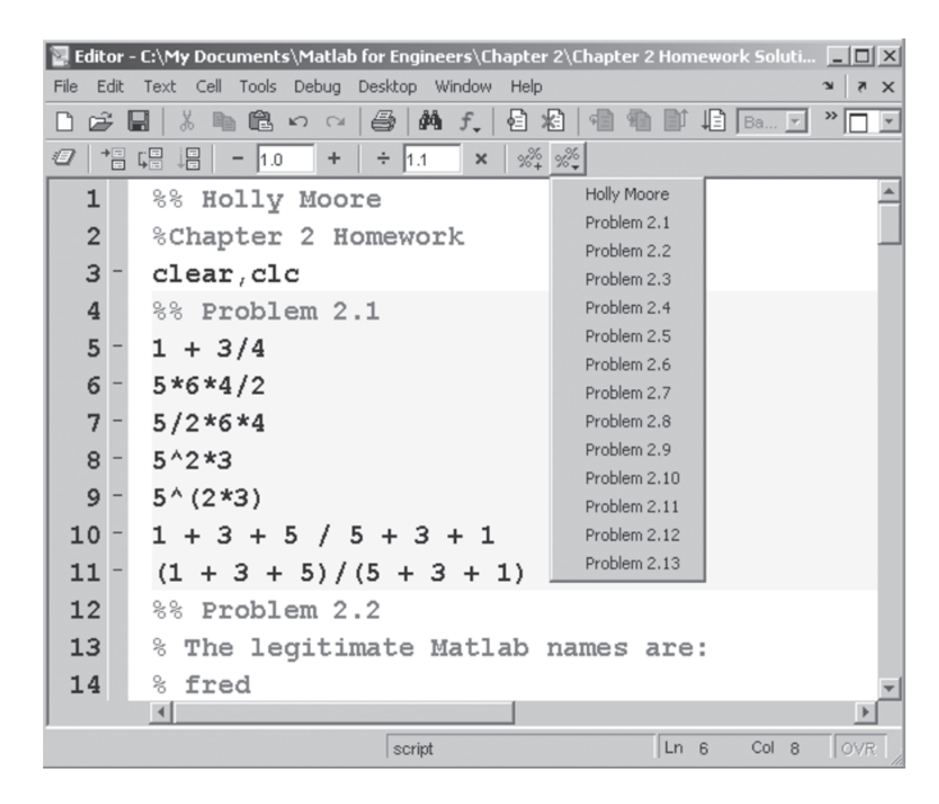
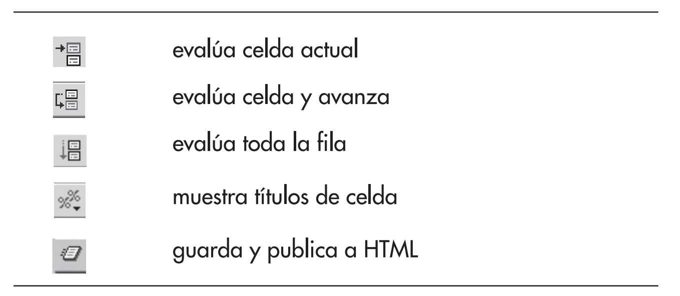


Figura 7.6
El icono Show Cell Titles (mostrar títulos de celda) menciona todas las celdas en el archivo-m.

Tabla 7.3 Barra de herramientas de celda



	evalúa celda actual
	evalúa celda y avanza
	evalúa toda la fila
	muestra títulos de celda
	guarda y publica a HTML

celda: sección de código MATLAB ubicado entre divisores de celda (%%)

La figura 7.6 muestra las primeras 14 líneas de un archivo-m escrito para resolver algunos problemas de tarea. Al dividir el programa en celdas fue posible trabajar en cada problema por separado. Asegúrese de guardar cualquier archivo-m que desarrolle de esta forma al seleccionar **Save** o **Save As** del menú archivo:

File → Save o

File → Save As

La razón para usar estos comandos es que el programa no se guarda automáticamente cada vez que lo corre.

Dividir en celdas un archivo-m de tarea ofrece una gran ventaja a quien deba calificar el ensayo. Al usar la función **evaluar celda y avanzar**, el calificador puede pasar un problema a la vez a través del programa.

La barra de herramientas de celda también permite al usuario publicar un programa archivo-m a un archivo HTML. MATLAB corre el programa y crea un reporte que muestra el código en cada celda, así como los resultados del cálculo que se enviaron a la ventana de

Idea clave: el modo celda le permite crear reportes en HTML, Word y PowerPoint.

comandos. Cualquier figura creada también se incluye en el reporte. La primera porción del reporte creado a partir del archivo-m de la figura 7.6 se muestra en la figura 7.7. Si prefiere un reporte en un formato diferente, como Word o PowerPoint, puede usar la opción de barra de menú

File → Publish To

para enviar los resultados en su elección de muchos formatos diferentes.

Finalmente, la barra de herramientas de celda incluye un conjunto de herramientas de manipulación de valor, como se muestra en la figura 7.8.

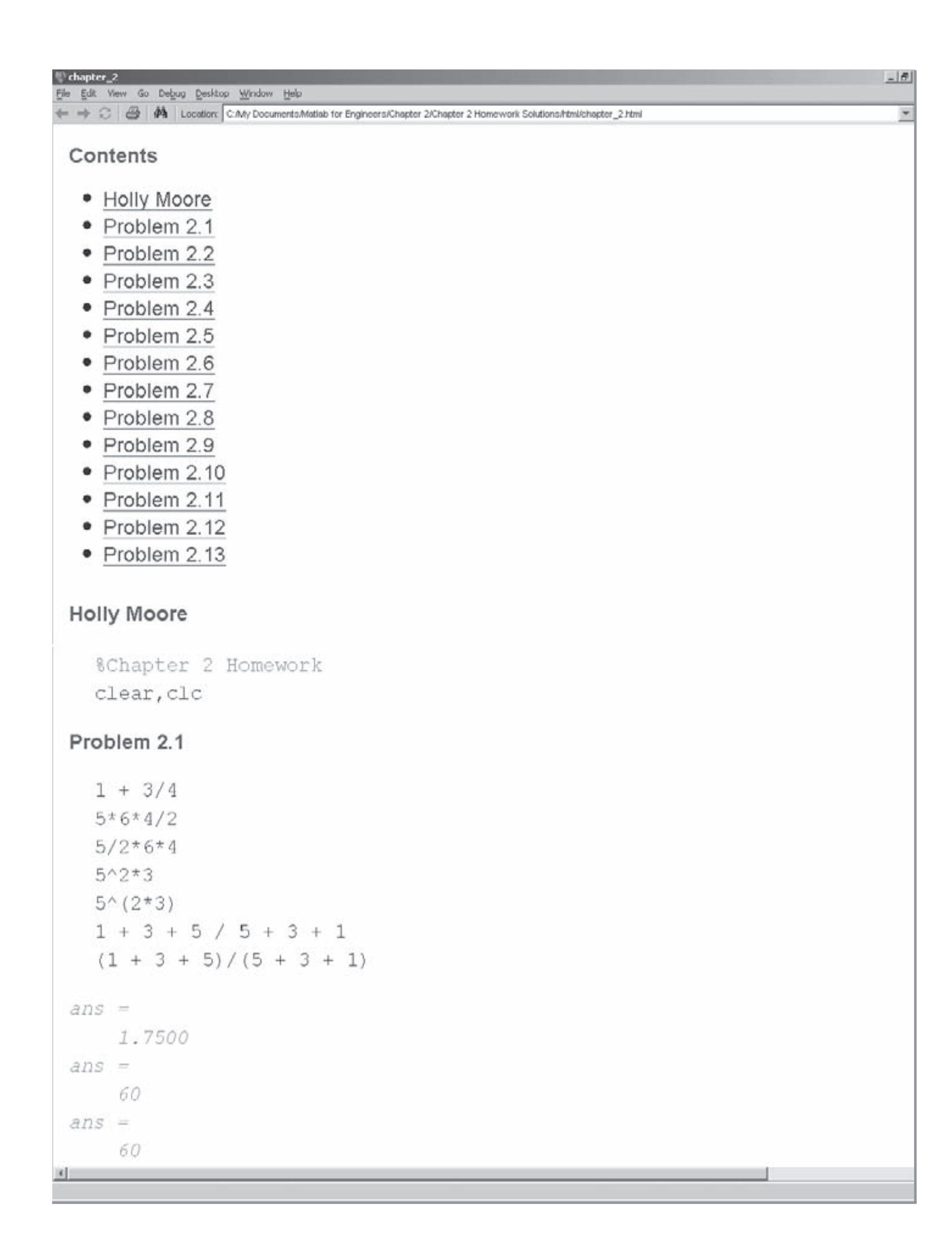


Figura 7.7
Reporte HTML creado a partir de un archivo-m MATLAB.

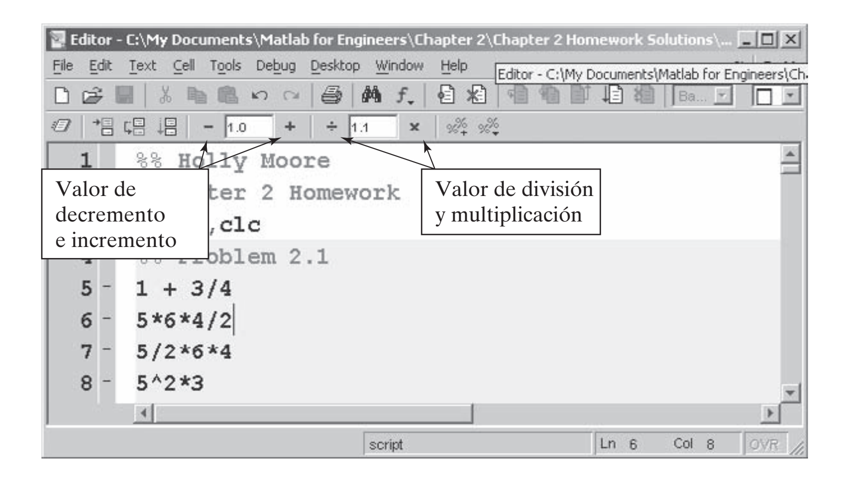


Figura 7.8
Las herramientas de manipulación de valor permiten al usuario experimentar con diferentes valores en los cálculos.

Cualquier número que esté más cerca del cursor (en la figura 7.8 es el número 2) se puede ajustar por el factor que se muestra en la barra de herramientas al seleccionar el icono apropiado (−, +, ÷ o ×). Cuando esta característica se usa en combinación con la herramienta **evaluar celda**, puede repetir un conjunto de cálculos muchas veces mientras ajusta fácilmente una variable de interés.

EJEMPLO 7.3

Ajuste interactivo de parámetros

Sobre la base de un cálculo de equilibrio de energía, usted sabe que el cambio en entalpía de una muestra de 1 kmol (29 kg) de aire que va del estado 1 al estado 2 es 8900 kJ. Usted quisiera saber la temperatura final, pero la ecuación que relaciona el cambio en entalpía a temperatura, a saber,

$$\Delta h = \int_1^2 C_p \, dT$$

donde

$$C_p = a + bT + cT^2 + dT^3$$

es muy complicada de resolver para la temperatura final. Sin embargo, al usar técnicas aprendida en cálculo, se encuentra que

$$\Delta h = a(T_2 - T_1) + \frac{b}{2}(T_2^2 - T_1^2) + \frac{c}{3}(T_2^3 - T_1^3) + \frac{d}{4}(T_2^4 - T_1^4)$$

Si se conoce la temperatura inicial (T_1) y los valores de a, b, c y d, se pueden adivinar valores para la temperatura final (T_2) hasta obtener el valor correcto de Δh. La habilidad interactiva para modificar valores de variable en el modo celda hace sencilla la resolución de este problema.

1. Establezca el problema.
   Encontrar la temperatura final del aire cuando usted conoce la temperatura inicial y el cambio en energía interna.

2. Describa las entradas y salidas.

Entrada Usados en la ecuación para C_p, los siguientes valores de a, b, c y d le darán un valor de capacidad calorífica en kJ/kmol K:

$$a = 28.90$$
$$b = 0.1967 \times 10^{-2}$$
$$c = 0.4802 \times 10^{-5}$$
$$d = -1.966 \times 10^{-9}$$
$$\Delta h = 8900 \text{ kJ}$$
$$T_1 = 300 \text{ K}$$

Salida Para cada valor supuesto de la temperatura final, en la pantalla se debe imprimir una estimación de Δh

3. Desarrolle un ejemplo a mano.
Si supone una temperatura final de 400 K, entonces

$$\Delta h = a(T_2 - T_1) + \frac{b}{2}(T_2^2 - T_1^2) + \frac{c}{3}(T_2^3 - T_1^3) + \frac{d}{4}(T_2^4 - T_1^4)$$

$$\Delta h = 28.9(400 - 300) + \frac{0.1967 \cdot 10^{-2}}{2}(400^2 - 300^2) + \frac{0.4802 \cdot 10^{-5}}{3}$$

$$\times (400^3 - 300^3) + \cdots \frac{-1.966 \cdot 10^{-9}}{4}(400^4 - 300^4)$$

que produce

$$\Delta h = 3009.47$$

4. Desarrolle una solución MATLAB.

```
%% Ejemplo 7.3
% Ajuste interactivo de parámetros
clear,clc
a = 28.90;
b = 0.1967e-2;
c = 0.4802e-5;
d = -1.966e-9;
T1 = 300
%% suponga T2 y ajuste
T2 = 400
format bank
delta_h = a*(T2-T1) + b*(T2.^2 - T1.^2)/2 + c*(T2.^3-
T1.^3)/3 + d*(T2.^4-T1.^4)/4
```

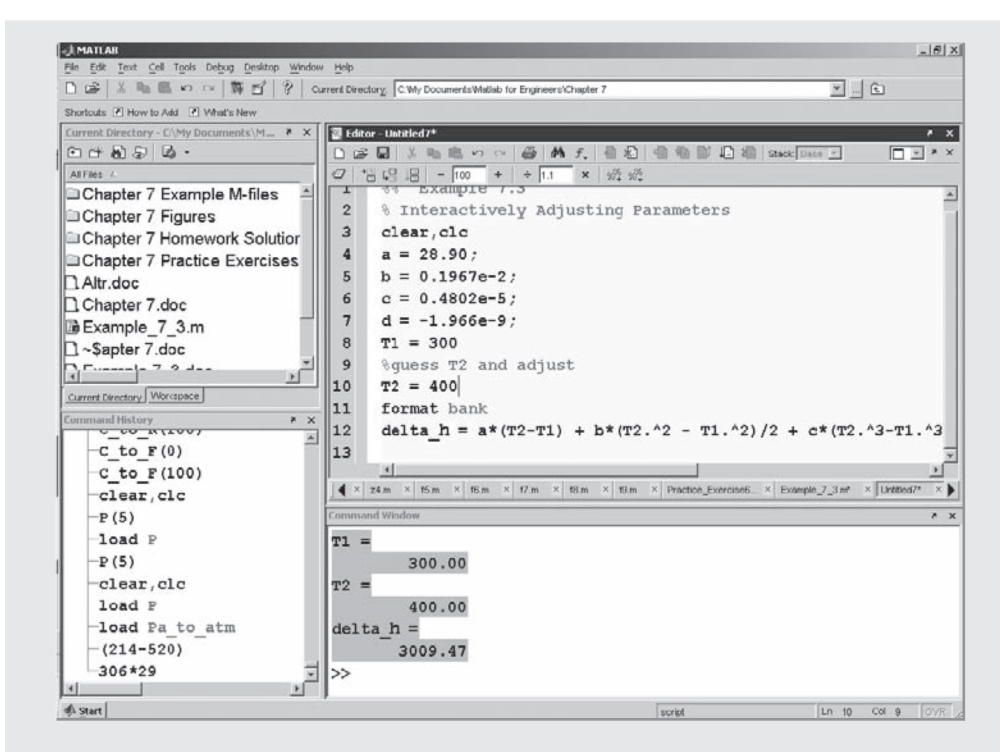


Figura 7.9
La suposición original le da una idea de cuán alejado está de la respuesta final.

Corra el programa una vez y MATLAB regresa

$$\texttt{T1} = 300.00$$
$$\texttt{T2} = 400.00$$
$$\texttt{delta_h} = 3009.47$$

Ahora coloque el cursor cerca del enunciado **T2 = 400**, como se muestra en la figura 7.9. (En este ejemplo, la ventana de edición se acopló con el escritorio MATLAB.) Al seleccionar el icono Increment Value, con el valor establecido a 100, se pueden intentar rápidamente muchas temperaturas diferentes. (Véase la figura 7.10.) Una vez que cierra, puede cambiar el incremento y el cero en la respuesta.

Un valor T_2 de 592 K dio un valor Δh calculado de 8927, que está bastante cerca de la meta. Se podría acercar más si creyera que la precisión añadida estaba justificada.

5. Ponga a prueba la solución.
   Sustituya el valor calculado de T_2 en la ecuación original y verifique los resultados con una calculadora:

$$\Delta h = 28.9(592 - 300) + \frac{0.1967 \cdot 10^{-2}}{2}(592^2 - 300^2) +$$

$$+ \frac{0.4802 \cdot 10^{-5}}{3}(592^3 - 300^3) + \frac{-1.966 \cdot 10^{-9}}{4}(592^4 - 300^4)$$

$$\Delta h = 8927.46$$

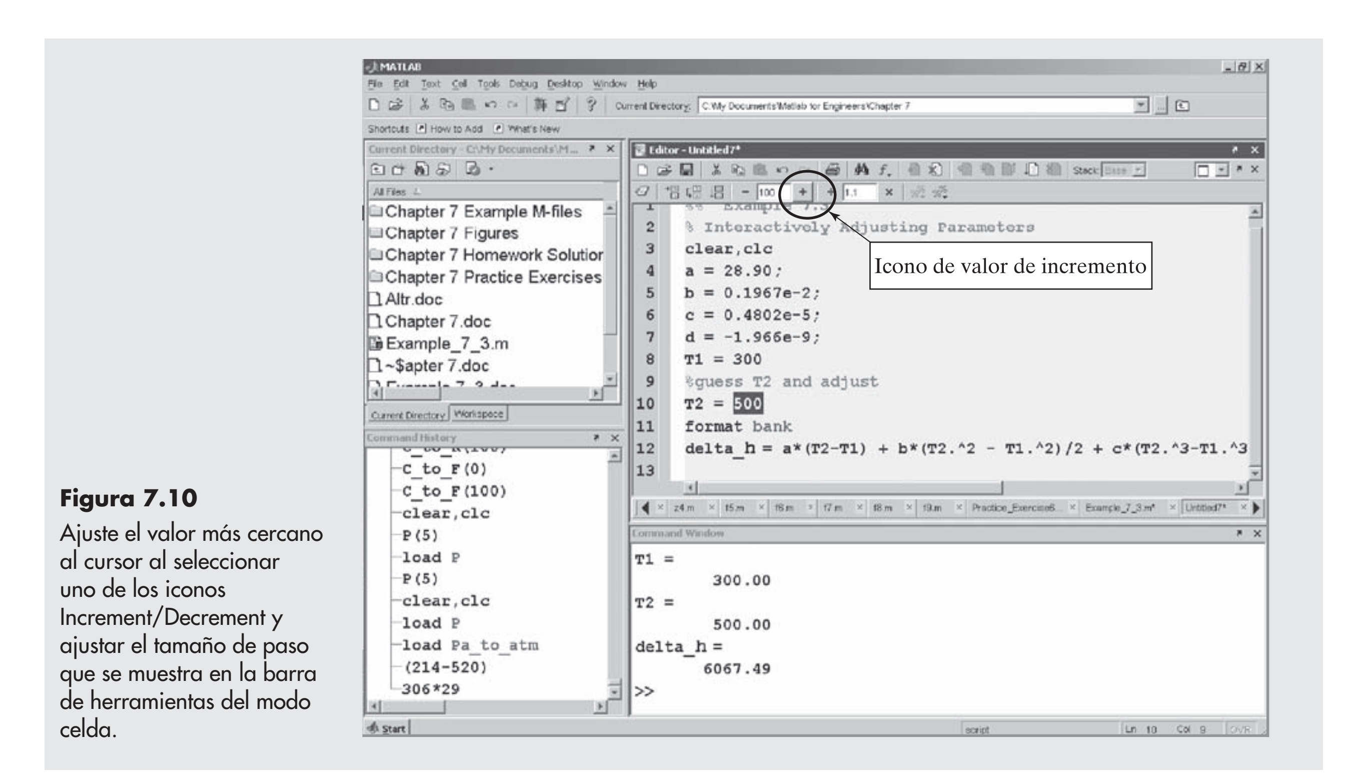


Figura 7.10
Ajuste el valor más cercano al cursor al seleccionar uno de los iconos Increment/Decrement y ajustar el tamaño de paso que se muestra en la barra de herramientas del modo celda.

7.5 LECTURA Y ESCRITURA DE DATOS DESDE ARCHIVOS

Idea clave: MATLAB puede importar datos de archivos que usan diversos formatos.

Los datos se almacenan en muchos formatos diferentes, dependiendo de los dispositivos y programas que crean los datos y de la aplicación. Por ejemplo, el sonido se puede almacenar en un archivo .wav, y una imagen se puede almacenar en un archivo .jpg. Muchas aplicaciones almacenan datos en hojas de cálculo Excel (archivos .xls). El más genérico de estos archivos es el archivo ASCII, que usualmente se almacena como un archivo .dat o .txt. Quizá usted quiera importar estos datos en MATLAB para analizar en un programa MATLAB, o tal vez quiera guardar sus datos en uno de estos formatos para hacer el archivo más fácil de exportar a otra aplicación.

7.5.1 Importación de datos

Importa Wizard (asistente de importación)

Si usted selecciona un archivo de datos del directorio actual y hace doble clic en el nombre del archivo, se lanza el Import Wizard (asistente de importación). El Import Wizard determina qué tipo de datos hay en el archivo y sugiere formas de representar los datos en MATLAB. La tabla 7.4 es una lista de algunos de los tipos de datos reconocidos por MATLAB. MATLAB no soporta todos los posibles formatos de datos. Puede encontrar una lista completa al escribir

```
doc fileformats
```

en la ventana de comandos.

Tabla 7.4 Tipos de archivo de datos soportados por MATLAB

Tipo de archivo	Extensión	Observación
Texto	.mat .dat .txt	área de trabajo MATLAB datos ASCII datos ASCII
Otros formatos comunes de datos científicos	.cdf .fits .hdf	formato de datos común datos en sistema flexible de transporte de imagen formato de datos jerárquicos
Datos de hoja de cálculo	.xls .wk1 .tiff	hoja de cálculo Excel Lotus 123 formato de archivo de imagen etiquetado
Datos de imagen	.bmp .jpeg o jpg .gif	mapa de bits grupo experto fotográfico unido formato de intercambio gráfico
Datos de audio	.au .wav	audio archivo wave Microsoft
Película	.avi	archivo intercalado audio/video

El Import Wizard se puede usar para archivos ASCII simples y para archivos de hoja de cálculo Excel. También puede lanzar el Import Wizard desde la línea de comando, con la función **uiimport**:

```
uiimport(' nombre de archivo.extensión ')
```

Por ejemplo, para importar el archivo de sonido **decision.wav**, escriba

```
uiimport(' decision.wav ')
```

Entonces se abre el Importa Wizard, como se muestra en la figura 7.11.

Cualquier técnica para lanzar el Importa Wizard requiere una interacción con el usuario (a través del Wizard). Si quiere cargar un archivo de datos desde un programa MATLAB, necesitará un enfoque diferente.

Comandos de importación

Puede evitar las interacciones del Wizard con una de las funciones que están especialmente diseñadas para leer cada uno de los formatos de archivo soportados. Por ejemplo, para leer un archivo .wav, use la función **wavread**:

```
[data,fs]=wavread('decision.wav')
```

Claramente, necesita entender qué tipo de datos esperar, de modo que pueda nombrar apropiadamente las variables creadas. Puede encontrar una lista de funciones de importación al escribir

```
doc fileformats
```

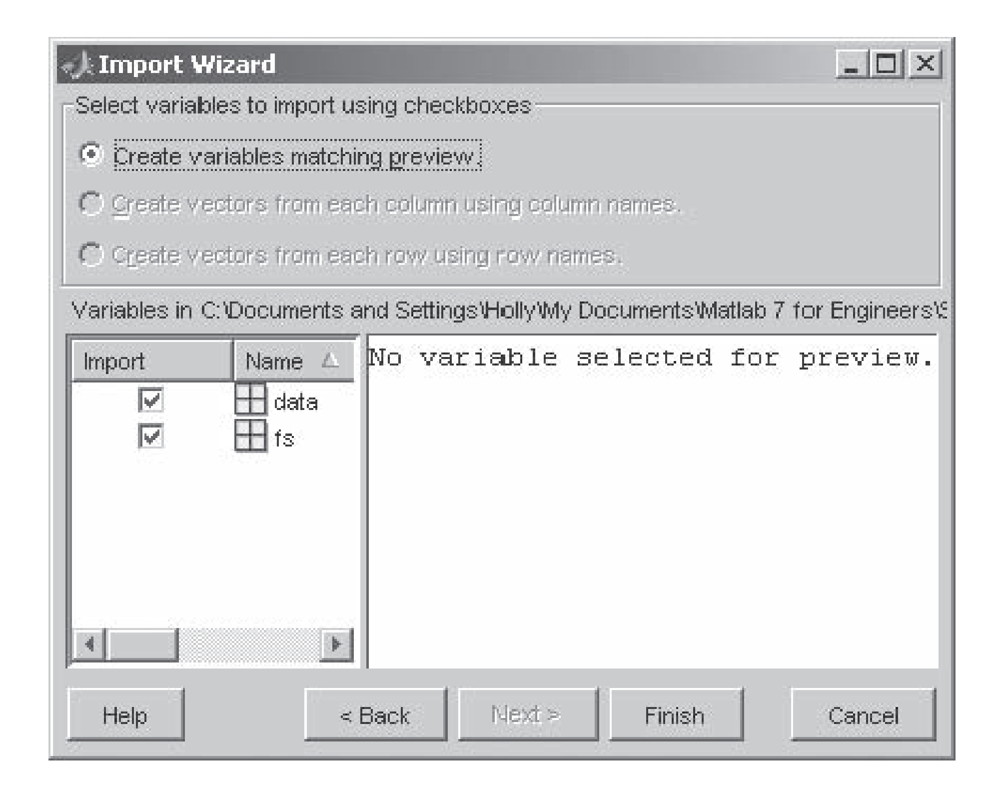


Figura 7.11
El Import Wizard se lanza cuando se ejecuta el comando **uiimport**.

EJEMPLO 7.4

2001: una odisea espacial: archivos de sonido

Uno de los personajes más memorables de la película *2001: Odisea del espacio* fue la computadora Hal. Los sonidos del diálogo de Hal en la película han sido populares durante años entre los programadores de computadoras y los ingenieros que usan computadoras. Usted puede encontrar archivos .wav de algunos de los diálogos de Hal en http://www.palantir.net/2001/ y en otros websites de cultura popular. Inserte los comentarios de Hal en un programa MATLAB. (Necesitará la función **sound**; consulte el tutorial **help** para detalles acerca de su uso.)

1. Establezca el problema.
   Cargar archivos de sonido en un programa MATLAB y reprodúzcalos en momentos apropiados.
2. Describa las entradas y salidas.

   Entrada Archivos de sonidos descargados de Internet. Para este ejemplo, se supondrá que descargó los siguientes tres archivos:

   dave.wav
   error.wav
   sure.wav

   Salida Reproducir los archivos de sonido dentro de un programa MATLAB
3. Desarrolle un ejemplo a mano.
   Aunque trabajar un ejemplo a mano no es apropiado para este problema, puede escuchar los archivos de sonido de Internet antes de insertarlos en el programa.

4. Desarrolle una solución MATLAB.
   Descargue los archivos de sonido y guárdelos en el directorio actual antes de correr el siguiente programa:

```
%% Ejemplo 7.4
% Archivos de sonido
%% Primer clip
[dave,fs_dave]=wavread('dave.wav');
disp('Presione enter una vez que el clip de sonido deje de tocar')
sound(dave,fs_dave)
pause
%% Segundo clip
[error,fs_error]=wavread('error.wav');
disp('Presione enter una vez que el clip de sonido deje de tocar')
sound(error,fs_error)
pause
%% Tercer clip
[sure,fs_sure]=wavread('sure.wav');
disp('Presione enter una vez que el clip de sonido deje de tocar')
sound(sure,fs_sure)
pause
disp('Ése fue el último clip')
```

5. Ponga prueba la solución.
   Existen muchos archivos de audio disponibles para descargar de Internet. Muchos son tan simples como éstos, pero algunos son piezas musicales completas. Navegue en Internet e inserte un "byte sonoro" en otro programa MATLAB, como un mensaje de error para sus usuarios. Algunos de los favoritos del autor son *Star Trek* (intente http://services.tos.net/sounds/sound.html#tos) y *Los Simpsons*.

7.5.2 Exportación de datos

La forma más sencilla de encontrar la función adecuada para escribir un archivo es usar el tutorial **help** para encontrar la función correcta para leerla y luego seguir las ligas hacia la función **write**. Por ejemplo, para leer un archivo de hoja de cálculo Excel (.xls), se usaría **xlsread**:

```
xlsread('nombre de archivo.xls')
```

Al final de la página tutorial se hace referencia a la función correcta para escribir un archivo Excel, a saber,

```
xlswrite('nombre de archivo.xls', M)
```

donde **M** es el arreglo que quiere almacenar en la hoja de cálculo Excel.

RESUMEN

MATLAB proporciona funciones que permiten al usuario interactuar con un programa archivo-m y que permiten al programador controlar la salida a la ventana de comandos.

La función **input** pausa el programa y envía una incitación determinada por el programador a la ventana de comandos. Una vez que el usuario ingresó un valor o valores y oprime la tecla return, continúa la ejecución del programa.

El comando display (**disp**) permite al programador desplegar los contenidos de una cadena o una matriz en la ventana de comandos. Aunque el comando **disp** es adecuado para muchas tareas de despliegue, el comando **fprintf** da al programador considerablemente más control sobre la forma en que los resultados se despliegan en la ventana de comandos. Ello permite al programador combinar texto y resultados calculados en la misma línea y especificar el número de formato usado.

Para aplicaciones en las que se requiere entrada gráfica, el comando **ginput** permite al usuario proporcionar entrada a un programa al seleccionar puntos de una ventana de gráficos.

El modo celda permite al programador agrupar código de archivo-m en secciones y correr cada sección de manera individual. La herramienta **publish to HTML** crea un reporte que contiene tanto el código del archivo-m y los resultados, así como cualesquiera figuras generadas cuando se ejecutó el programa. Los iconos Incremento y Decremento en la barra de herramientas de celda permiten al usuario cambiar automáticamente el valor de un parámetro cada vez que el código se ejecuta, lo que hace sencillo probar el resultado de cambiar una variable.

MATLAB incluye funciones que permiten al usuario importar y exportar datos en algunos formatos de archivo populares. Una lista completa de dichos formatos está disponible en el tutorial **help** en la página File Formats (doc fileformats).

RESUMEN MATLAB

El siguiente resumen MATLAB menciona todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres especiales	
'	comienza y termina una cadena
%	marcador de posición usado en el comando **fprintf**
%f	notación punto fijo, o decimal
%e	notación exponencial
%g	notación o punto fijo o exponencial
%s	notación cadena
%%	divisor de celda
\n	salto de línea (linefeed)
\r	regreso de carro (similar a linefeed)
\t	tabulador
\b	retroceder un espacio (backspace)

Comandos y funciones	
disp	despliega una cadena o una matriz en la ventana de comandos
fprintf	controla el despliegue de ventana de comandos
ginput	permite al usuario elegir valores de una gráfica
input	permite al usuario ingresar valores
num2str	cambia un número a una cadena
pause	pausa el programa
sound	reproduce datos MATLAB a través de las bocinas
uiimport	lanza el Importa Wizard
wavread	lee archivos wave
xlsimport	importa archivos de datos Excel
xlswrite	exporta datos como un archivo Excel

TÉRMINOS CLAVE

ancho de campo
arreglo carácter
cadena
campo precisión
celda
modo celda
salida formateada

PROBLEMAS

Función input

7.1 Cree un archivo-m que conmine al usuario a ingresar un valor de *x* y luego calcule el valor de sen(*x*).

7.2 Cree un archivo-m que conmine al usuario a ingresar una matriz y luego use la función **max** para determinar el valor ingresado más grande. Use la siguiente matriz para probar su programa:

[1,5,3,8,9,22]

7.3 El volumen de un cono es

$$V = \frac{1}{3} \times \text{área_de_la_base} \times \text{altura}$$

Conmine al usuario a ingresar el área de la base y la altura del cono (figura P7.3). Calcule el volumen del cono.

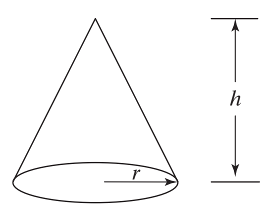


Figura P7.3
Volumen de un cono.

Función disp

7.4 Uno de los primeros programas de cómputo que muchos estudiantes escriben se llama “Hola, mundo”. Lo único que hace el programa es imprimir este mensaje en la pantalla de la computadora. Escriba un programa “Hola, mundo” en un archivo-m con la función **disp**.

7.5 Use dos enunciados **input** separados para conminar al usuario a ingresar su nombre y apellidos. Use la función **disp** para desplegar dichos nombres en una línea. (Necesitará combinar los nombres y algunos espacios en un arreglo.)

7.6 Conmine al usuario a ingresar su edad. Luego use la función **disp** para reportar la edad de vuelta a la ventana de comandos. Si, por ejemplo, el usuario ingresa 5 cuando se le solicita la edad, su despliegue debe leerse

```
Su edad es 5
```

Esta salida requiere combinar tanto datos carácter (una cadena) como datos numéricos en la función **disp**, lo que se puede lograr al usar la función **num2str**.

7.7 Conmine al usuario a ingresar un arreglo de números. Use la función **length** para determinar cuántos valores ingresó y use la función **disp** para reportar sus resultados a la ventana de comandos.

fprintf

7.8 Repita el problema 7.7 y use **fprintf** para reportar sus resultados.

7.9 Use **fprintf** para crear las tablas de multiplicación de 1 a 13 para el número 6. Su tabla se debe ver como esto

1 por 6 es 6
2 por 6 es 12
3 por 6 es 18
⋮

7.10 Antes de que las calculadoras fueran fácilmente asequibles (alrededor de 1974), los estudiantes usaban tablas para determinar los valores de las funciones matemáticas como seno, coseno y log. Cree una de tales tablas para seno, con los siguientes pasos:

- Cree un vector de valores ángulo desde 0 hasta 2π en incrementos de $\pi/10$.
- Calcule el seno de cada uno de los ángulos y agrupe sus resultados en una tabla que incluya el ángulo y el seno.
- Use **disp** para crear un título para la tabla y un segundo comando **disp** para crear encabezados de columna.
- Use la función **fprintf** para desplegar los números. Despliegue sólo dos valores después del punto decimal.

7.11 Las dimensiones muy pequeñas, las que están a escala atómica, con frecuencia se miden en angstroms. El símbolo para un angstrom es Å y corresponde a una longitud de 10^{-10} metros. Cree una tabla de conversión de pulgadas a angstroms del modo siguiente, para valores de pulgadas desde 1 hasta 10:

- Use **disp** para crear un título y encabezados de columna.
- Use **fprintf** para desplegar la información numérica.
- Puesto que la longitud representada en angstroms es demasiado grande, represente su resultado en notación científica y muestre dos valores después del punto decimal. Esto corresponde a tres cifras significativas (una antes y dos después del punto decimal).

7.12 Use su buscador favorito de Internet y navegue la red para identificar conversiones monetarias recientes para libras esterlinas británicas, yen japonés y el euro europeo a dólares estadounidenses. Use las tablas de conversión para crear las siguientes tablas (use los comandos **disp** y **fprintf** en su solución, que debe incluir un título, etiquetas de columna y salida formateada):

(a) Genere una tabla de conversiones de yen a dólar. Comience la columna yen en 5 e incremente por 5 yen. Imprima 25 líneas en la tabla.
(b) Genere una tabla de conversiones de euros a dólares. Comience la columna euro en 1 euro e incremente por 2 euros. Imprima 30 líneas en al tabla.
(c) Genere una tabla con cuatro columnas. La primera debe contener dólares, la segunda debe contener el número equivalente de euros, la tercera el número equivalente de libras y la cuarta el número equivalente de yen. Haga que la columna dólar varíe de 1 a 10.

Problemas que combinan los comandos input, disp y fprintf

7.13 Este problema requiere que usted genere tablas de conversión de temperatura. Use las siguientes ecuaciones, que describen las relaciones entre temperaturas en grados Fahrenheit (T_F), grados Celsius (T_C), grados Kelvin (T_K) y grados Rankine (T_R), respectivamente:

$$T_F = T_R - 459.67\ °R$$

$$T_F = \frac{9}{5}T_C + 32\ °F$$

$$T_R = \frac{9}{5}T_K$$

¡Necesitará reordenar estas expresiones para resolver algunos de los problemas!

(a) Genere una tabla de conversiones de Fahrenheit a Kelvin para valores desde 0 °F hasta 200 °F. Permita que el usuario ingrese los incrementos en grados F entre líneas. Use **disp** y **fprintf** para crear una tabla con un título, encabezados de columna y espaciamiento adecuado.

(b) Genere una tabla de conversiones de Celsius a Rankine. Permite que el usuario ingrese la temperatura inicial y los incrementos entre líneas. Imprima 25 líneas en la tabla. Use **disp** y **fprintf** para crear una tabla con un título, encabezados de columna y espaciamiento apropiado.

(c) Genere una tabla de conversiones de Celsius a Fahrenheit. Permita que el usuario ingrese la temperatura inicial, el incremento entre líneas y el número de líneas para la tabla. Use **disp** y **fprintf** para crear una tabla con un título, encabezados de columna y espaciamiento apropiado.

7.14 Los ingenieros usan regularmente unidades tanto inglesas como SI (Système International d'Unités). Algunos campos usan principalmente uno u otro, pero muchos combinan los dos sistemas. Por ejemplo, la tasa de entrada de energía a una planta de potencia de vapor a partir de la quema de combustibles fósiles usualmente se mide en Btu/hora. Sin embargo, la electricidad producida por la misma planta, por lo general, se mide en joules/s (watts). En contraste, los motores de automóvil con frecuencia se califican en caballos de fuerza o en pie lb_f/s. He aquí algunos factores de conversión que relacionan estas diferentes mediciones de potencia:

$$1 \text{ kW} = 3412.14 \text{ Btu/h} = 737.56 \text{ ft lb}_f\text{/s}$$

$$1 \text{ hp} = 550 \text{ ft lb}_f\text{/s} = 2544.5 \text{ Btu/h}$$

(a) Genere una tabla de conversiones de kW a hp. La tabla debe comenzar en 0 kW y terminar en 15 kW. Use la función **input** para permitir al usuario definir el incremento entre entradas de la tabla. Use **disp** y **fprintf** para crear una tabla con un título, encabezados de columna y espaciamiento apropiado.

(b) Genere una tabla de conversiones de ft lb_f/s a Btu/h. La tabla debe comenzar en 0 kW, pero permitir al usuario definir el incremento entre entradas de tabla y el valor final de la tabla. Use **disp** y **fprintf** para crear una tabla con un título, encabezados de columna y espaciamiento apropiado.

(c) Genere una tabla que incluya conversiones de kW a Btu/h y ft lb_f/s. Permita al usuario definir el valor inicial de kW, el valor final de kW y el número de entradas en la tabla. Use **disp** y **fprintf** para crear una tabla con un título, encabezados de columna y espaciamiento apropiado.

ginput

7.15 En el tiempo $t = 0$, el motor de un cohete se apaga con el cohete habiendo alcanzado una altitud de 500 metros y elevándose con una rapidez de 125 metros por segundo. En este punto, la gravedad toma el control. La altura del cohete como función del tiempo es

$$h(t) = -\frac{9.8}{2}t^2 + 125t + 500 \text{ para } t > 0$$

Grafique la altura del cohete desde 0 hasta 30 segundos y

- Use la función **ginput** para estimar la altura máxima que el cohete alcanza y el tiempo cuando el cohete golpea el suelo.
- Use el comando **disp** para reportar sus resultados a la ventana de comandos.

7.16 La función **ginput** es útil para escoger distancias de una gráfica. Demuestre esta característica mediante la realización de lo siguiente:

- Cree una gráfica de un círculo mediante la definición de un arreglo de ángulos desde 0 hasta 2 π, con un espaciamiento de $\pi/100$.
- Use la función **ginput** para elegir dos puntos en la circunferencia del círculo.

- Use **hold on** para evitar que la figura se refresque y grafique una línea entre los dos puntos que eligió.
- Use los datos de los puntos para calcular la longitud de la línea entre ellos. (*Sugerencia:* use el teorema de Pitágoras en su cálculo.)

Modo celda

7.17 Cree un archivo-m que contenga sus soluciones a los problemas de tarea de este capítulo. Use divisores de celda (%%) para dividir su programa en celdas (secciones) y titule cada sección con un número de problema. Corra su programa con la característica **evaluar celda y avanzar** de la barra de herramientas de celda.

7.18 Publique su programa y resultados del problema 7.17 en HTML, con la característica **publish to HTML** de la barra de herramientas de celda. Desafortunadamente, dado que la tarea de este capítulo requiere interacción con el usuario, los resultados publicados incluirán errores.



CAPÍTULO 8

Funciones lógicas y estructuras de control

Objetivos

Después de leer este capítulo, el alumno será capaz de

- entender cómo MATLAB interpreta los operadores relacionales y lógicos.
- usar la función **find**.
- comprender los usos adecuados de la familia de comandos **if/else**.
- comprender la estructura **switch/case**.
- escribir y usar bucles **for** y **while**.

INTRODUCCIÓN

Una forma de pensar los programas de cómputo (no sólo MATLAB) es considerar cómo se organizan los enunciados que componen el programa. Por lo general, las secciones del código de los programas de cómputo se pueden categorizar en una de tres estructuras: ***secuencias***, ***estructuras de selección*** y ***estructuras de repetición***. (Véase la figura 8.1.) Hasta el momento, se ha escrito código que contiene secuencias, pero ninguna de las otras estructuras:

- Las secuencias son listas de comandos que se ejecutan una después de otra.
- Una estructura de selección permite al programador ejecutar un comando (o conjunto de comandos) si algún criterio es verdadero, y un segundo comando o conjunto de comandos si el criterio es falso. Un enunciado de selección proporciona los medios de elegir entre dichas rutas, con base en una ***condición lógica***. Las condiciones que se evalúan con frecuencia contienen operadores tanto ***relacionales*** como ***lógicos*** o funciones.
- Una estructura de repetición, o bucle, hace que un grupo de enunciados se ejecute varias veces. El número de veces que se ejecuta un bucle depende de un contador o de la evaluación de una condición lógica.

8.1 OPERADORES RELACIONALES Y LÓGICOS

Las estructuras de selección y repetición que se usan en MATLAB dependen de operadores relacionales y lógicos. MATLAB tiene seis operadores relacionales para comparar dos matrices de igual tamaño, como se muestra en la tabla 8.1.

Las comparaciones son verdaderas o falsas, y la mayoría de los programas de cómputo (incluido MATLAB) usa el número 1 para verdadero (true) y el 0 para falso (false). (En realidad, MATLAB toma cualquier número distinto de cero como verdadero.) Si se definen dos escalares

```
x = 5;
y = 1;
```

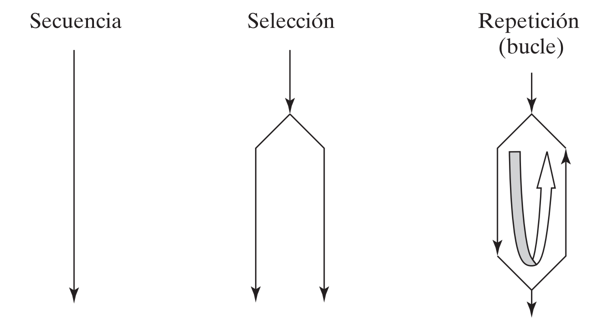


Figura 8.1
Estructuras de programación usadas en MATLAB.

y usa un operador relacional como <, el resultado de la comparación

```
x<y
```

es verdadero o falso. En este caso, **x** no es menor que **y**, por lo que MATLAB responde

```
ans =
     0
```

lo que indica que la comparación fue falsa. MATLAB usa esta respuesta en enunciados de selección y en estructuras de repetición para tomar decisiones.

Desde luego, las variables en MATLAB, por lo general, representan matrices completas. Si se redefinen **x** y **y**, se puede ver cómo MATLAB maneja las comparaciones entre matrices. Por ejemplo,

```
x = 1:5;
y = x -4;
x<y
```

regresa

```
ans =
      0     0     0     0     0
```

MATLAB compara elementos correspondientes y crea una matriz respuesta de ceros y unos. En el ejemplo previo, **x** fue mayor que **y** para cada comparación de elementos, de modo que toda la comparación fue falsa y la respuesta fue una cadena de ceros. Si, en vez de ello, se tiene

```
x = [ 1, 2, 3, 4, 5];
y = [-2, 0, 2, 4, 6];
x<y
```

entonces

```
ans =
     0     0     0     0     1
```

El resultado indica que la comparación fue falsa para los primeros cuatro elementos, pero verdadera para el último. Para que una comparación sea verdadera para toda una matriz, debe ser verdadera para *cada* elemento en la matriz. En otras palabras, todos los resultados deben ser unos.

Idea clave: los operadores lógicos se usan para combinar enunciados de comparación.

MATLAB también le permite combinar comparaciones con los operadores lógicos ***and***, ***not*** y ***or***. (Véase la tabla 8.2.)

El código

```
x = [ 1, 2, 3, 4, 5];
y = [-2, 0, 2, 4, 6];
z = [ 8, 8, 8, 8, 8];
z>x & z>y
```

Tabla 8.1 Operadores relacionales

Operador relacional	Interpretación
<	menor que
<=	menor que o igual a
>	mayor que
>=	mayor que o igual a
==	igual a
~=	no igual a

Idea clave: los operadores relacionales comparan valores.

Tabla 8.2 Operadores lógicos

Operador lógico	Interpretación
&	and
~	not
|	or
xor	or exclusiva

regresa

```
ans =
     1     1     1     1     1
```

puesto que **z** es mayor que **x** y **y** para cada elemento. El enunciado

```
x>y | x>z
```

se lee como "**x** es mayor que **y** o **x** es mayor que **z**" y regresa

```
ans =
     1     1     1     0     0
```

Esto significa que la condición es verdadera para los primeros tres elementos y falsa para los últimos dos.

Estos operadores relacionales y lógicos se usan tanto en estructuras de selección como en bucles para determinar qué comandos se deben ejecutar.

8.2 DIAGRAMAS DE FLUJO Y SEUDOCÓDIGO

Con la adición de las estructuras de selección y las estructuras de repetición a su grupo de herramientas de programación se vuelve todavía más importante planear su programa antes de que comience a codificar. Dos enfoques comunes son: usar diagramas de flujo o usar seudocódigo. Los diagramas de flujo son un enfoque gráfico para crear su plan de codificación, y el seudocódigo es una descripción verbal de su plan. Tal vez quiera usar alguno o ambos para sus proyectos de programación.

Idea clave: los diagramas de flujo y el seudocódigo se usan para planear tareas de programación.

Para programas simples, el seudocódigo puede ser el mejor enfoque de planeación (o al menos el más simple):

- Resalte un conjunto de enunciados que describan los pasos que tomará para resolver un problema.
- Convierta estos pasos en comentarios en un archivo-m.
- Inserte el código MATLAB apropiado en el archivo entre las líneas de comentario.

He aquí un ejemplo realmente simple: suponga que se le pide crear un programar para convertir mph a pies/s. La salida debe ser una tabla completa con un título y encabezados de columna. He aquí una forma de resaltar los pasos que puede seguir:

- Definir un vector de valores mph.
- Convertir mph a pies/s.
- Combinar los vectores mph y pies/s en una matriz.
- Crear un título de tabla.
- Crear encabezados de columna.
- Desplegar la tabla.

Una vez que defina los pasos, póngalos en un archivo-m MATLAB como comentarios:

```
%Definir un vector de valores mph
%Convertir mph a pies/s
%Combinar los vectores mph y pies/s en una matriz
%Crear un título de tabla
%Crear encabezados de columna
%Desplegar la tabla
```

Ahora puede insertar el código MATLAB apropiado en el archivo-m

```
%Definir un vector de valores mph
  mph = 0:10:100;
%Convertir mph a pies/s
  fps = mph*5280/3600;
%Combinar los vectores mph y pies/s en una matriz
  table = [mph;fps]
%Crear un título de tabla
  disp('Tabla de conversión de velocidad')
%Crear encabezados de columna
  disp('   mph    f/s')
%Desplegar la tabla
  fprintf('%8.0f  %8.2f \n',table)
```

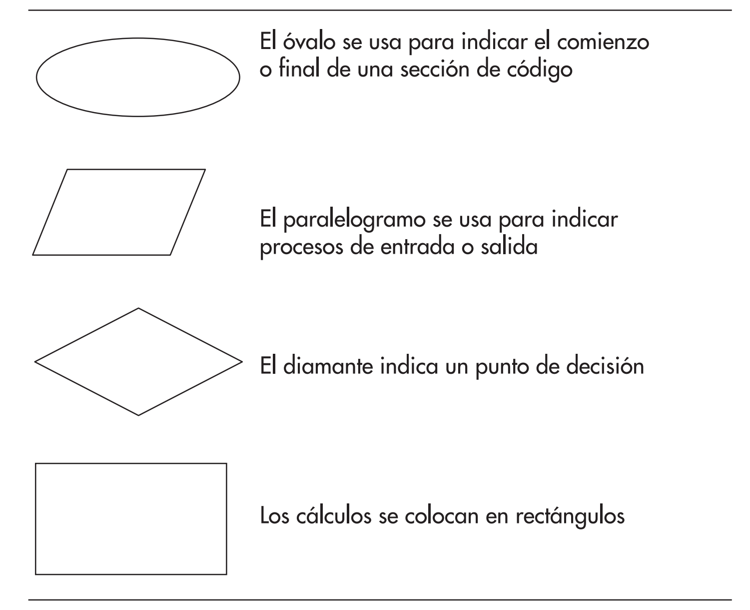
Si pone algo de tiempo en su planeación, probablemente no necesitará cambiar mucho el seudocódigo una vez que inicie la programación.

Los diagramas de flujo solos o combinados con seudocódigo son especialmente apropiados para tareas de programación más complicadas. Puede crear un "gran cuadro" de su programa gráficamente y luego convertir su proyecto a seudocódigo adecuado para ingresar en el programa como comentarios. Antes de comenzar a hacer diagramas de flujo, necesitará una introducción a algunos símbolos estándar de los diagramas de flujo. (Véase la tabla 8.3.)

La figura 8.2 es un ejemplo de un diagrama de flujo para el problema mph a pies/s. Para un problema así de simple, probablemente en realidad nunca crearía un diagrama de flujo. Sin embargo, conforme los problemas se vuelven más complicados, los diagramas de flujo se convierten en una herramienta invaluable que le permite organizar sus pensamientos.

Una vez que cree un diagrama de flujo debe transferir las ideas en líneas de comentario en un archivo-m y luego agregar el código apropiado entre los comentarios.

Tabla 8.3 Diagramas de flujo para diseñar programas de cómputo



	El óvalo se usa para indicar el comienzo o final de una sección de código
	El paralelogramo se usa para indicar procesos de entrada o salida
	El diamante indica un punto de decisión
	Los cálculos se colocan en rectángulos

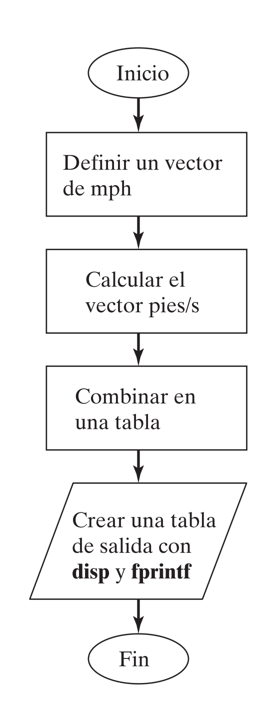


Figura 8.2
Los diagramas de flujo facilitan la visualización de la estructura de un programa.

Recuerde: tanto los diagramas de flujo como el seudocódigo son herramientas que tienen la intención de ayudarle a crear programas de cómputo. También se pueden usar de manera efectiva para ilustrar la estructura de un programa a los no programadores, dado que enfatizan la progresión lógica de ideas sobre los detalles de programación.

diagrama de flujo: representación pictórica de un programa de cómputo

seudocódigo: lista de tareas de programación necesarias para crear un programa

8.3 FUNCIONES LÓGICAS

MATLAB ofrece tanto estructuras de selección tradicionales por ejemplo la familia de funciones **if** como una serie de funciones lógicas que realizan en gran medida la misma tarea. La función lógica principal es **find**, que con frecuencia se puede usar en lugar tanto de las estructuras de selección tradicionales como de los bucles.

8.3.1 Find

El comando **find** busca una matriz e identifica cuáles elementos en dicha matriz satisfacen un criterio dado. Por ejemplo, la Academia Naval de Estados Unidos requiere solicitantes que tengan al menos 5′6″ (66″) de alto. Considere esta lista de alturas de solicitantes:

```
height = [63,67,65,72,69,78,75]
```

Puede encontrar los números índice de los elementos que satisfacen el criterio al usar el comando **find**:

```
accept = find(height>=66 )
```

Este comando regresa

```
accept =
   2   4   5   6   7
```

Idea clave: con frecuencia, las funciones lógicas son herramientas de programación más eficientes que las estructuras de selección tradicionales.

La función **find** regresa los números índice de la matriz que satisfacen el criterio. Si quiere saber cuáles son las estaturas reales, puede llamar a cada elemento con el número índice:

```
height(accept)
ans =
  67  72  69  78  75
```

También podría determinar cuáles solicitantes *no* satisfacen el criterio. Use

```
decline = find(height<66)
```

que produce

```
decline =
  1 3
```

Podría usar el comando **disp** y **fprintf** para crear un reporte más legible:

```
disp('Los siguientes candidatos satisfacen el requisito de
  estatura'); fprintf('Candidato # %4.0f mide %4.0f
  pulgadas de alto \n', [accept;height(accept)])
```

Estos comandos regresan la siguiente tabla en la ventana de comandos:

```
Los siguientes candidatos satisfacen el requisito de estatura
Candidato #    2 mide    67 pulgadas de alto
Candidato #    4 mide    72 pulgadas de alto
Candidato #    5 mide    69 pulgadas de alto
Candidato #    6 mide    78 pulgadas de alto
Candidato #    7 mide    75 pulgadas de alto
```

Obviamente, también podría crear una tabla de quienes no satisfacen el requisito:

```
disp('Los siguientes candidatos no satisfacen el requisito
  de estatura')
fprintf('Candidato # %4.0f mide %4.0f pulgadas de alto \n',
  [decline;height(decline)])
Candidato #    1 mide   63 pulgadas de alto
Candidato #    3 mide   65 pulgadas de alto
```

Puede crear criterios de búsqueda bastante complicados que usen los operadores lógicos. Por ejemplo, suponga que los solicitantes deben tener al menos 18 años de edad y menos de 35. Entonces sus datos se pueden ver como esto:

Estatura, pulgadas	Edad, años
63	18
67	19
65	18
72	20
69	36
78	34
75	12

Ahora defina la matriz y encuentre los números índice de los elementos en la columna 1 que sean mayores que 66. Entonces se encuentra cuáles de dichos elementos en la columna 2 son también mayores que o iguales a 18 y menores que o iguales a 35. Se usan los comandos

```
applicants = [   63, 18; 67, 19; 65, 18; 72, 20; 69, 36;
                 78, 34; 75, 12]
```

```
pass =find(applicants(:,1)>=66 & applicants(:,2)>=18
           & applicants(:,2) < 35)
```

que regresa

```
pass =
   2
   4
   6
```

la lista de solicitantes que satisfacen todos los criterios. Se podría usar **fprintf** para crear una salida más agradable. Primero cree una tabla de los datos a desplegar:

```
result = [pass,applicants(pass,1),applicants(pass,2)]';
```

Luego use **fprintf** para enviar los resultados a la ventana de comandos:

```
fprintf('Solicitante # %4.0f mide %4.0f pulgadas de alto
  y %4.0f años de edad\n',results)
```

La lista resultante es

```
Candidato #   2 mide  67 pulgadas de alto y  19 años de edad
Candidato #   4 mide  72 pulgadas de alto y  20 años de edad
Candidato #   6 mide  78 pulgadas de alto y  34 años de edad
```

Hasta el momento se usó **find** sólo para regresar un solo número índice. Si se definen dos salidas desde **find**, como en

```
[row, col] = find( criteria)
```

regresará los números de fila y columna adecuados (también llamados números índice o subíndices fila y columna).

Ahora, imagine que ya tiene una matriz de valores de temperatura de paciente medidos en una clínica. La columna representa el número de la estación donde se tomó la temperatura. Por tanto, el comando

```
temp = [95.3, 100.2, 98.6; 97.4,99.2, 98.9; 100.1,99.3, 97]
```

produce

```
temp =
  95.3000  100.2000   98.6000
  97.4000   99.2000   98.9000
  100.1000  99.3000   97.0000
```

y

```
element = find(temp>98.6)
```

produce el número de elemento para la representación de índice sencillo:

```
element =
      3
      4
      5
      6
      8
```

Cuando se usa el comando **find** con una matriz bidimensional, usa un esquema de numeración de elemento que opera en cada columna una a la vez. Por ejemplo, considere la matriz 3×3. Los números índice de elemento se muestran en la figura 8.3. Los elementos que contienen valores mayores que 98.6 se muestran en negrillas.

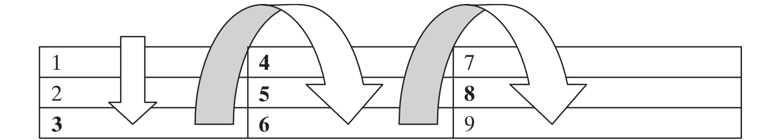


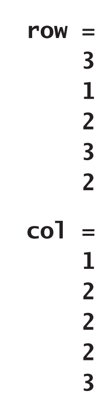
Figura 8.3
Secuencia de numeración de elementos para una matriz 3×3.

Con la finalidad de determinar los números de fila y columna, se necesita la sintaxis

```
[row, col] = find(temp>98.6)
```

que produce los siguientes números de fila y columna:

Idea clave: MATLAB es dominante en columna.



```
row =
    3
    1
    2
    3
    2
col =
    1
    2
    2
    2
    3
```

1, 1	**1, 2**	1, 3
2, 1	**2, 2**	**2, 3**
3, 1	**3, 2**	3, 3

Figura 8.4
Designación de fila, elemento para una matriz 3×3. Los elementos que satisfacen el criterio se muestran en negrillas.

En conjunto, estos números identifican los elementos que se muestran en la figura 8.4.

Al usar **fprintf** se puede crear un reporte más legible. Por ejemplo,

```
fprintf('Paciente%3.0f en estación%3.0f tuvo una temp de%6.
  1f \n', [row,col,temp(element)]')
```

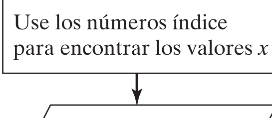
regresa

```
Paciente  3 en estación  1 tuvo una temp de 100.1
Paciente  1 en estación  2 tuvo una temp de 100.2
Paciente  2 en estación  2 tuvo una temp de  99.2
Paciente  3 en estación  2 tuvo una temp de  99.3
Paciente  2 en estación  3 tuvo una temp de  98.9
```

Inicio

Definir un vector de valores *x*

Encontrar los números índice en la matriz *x* para valores mayores que 9



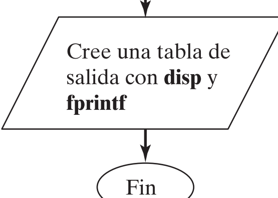


Figura 8.5
Diagrama de flujo que ilustra el comando **find**.

8.3.2 Diagrama de flujo y seudocódigo para comandos find

El comando **find** regresa sólo una respuesta: un vector de los números de elemento solicitados. Por ejemplo, puede hacer un diagrama de flujo de una secuencia de comandos, como se muestra en la figura 8.5. Si usa **find** muchas veces para separar una matriz en categorías, *puede* elegir emplear una forma diamante, que indica el uso de **find** como una estructura de selección.

```
%Defina un vector de valores x
 x=[1,2,3; 10, 5,1; 12,3,2;8, 3,1]
%Encuentre los números índice de los valores en x >9
 element = find(x>9)
%Use los números índice para encontrar los valores x
%mayores que 9 al ponerlos en x
 values = x(element)
```

```
% Cree una tabla de salida
 disp('Elementos mayores que 9')
 disp('Elemento #   Valor')
 fprintf('%8.0f    %3.0f \n', [element';values'])
```

EJEMPLO 8.1

Procesamiento de señal con el uso de la función sinc



Figura 8.6
Los osciloscopios se usan ampliamente en aplicaciones de procesamiento de señal. (Cortesía de Agilent Technologies Inc.)

La función sinc se usa en muchas aplicaciones de ingeniería, pero especialmente en procesamiento de señales (figura 8.6). Desafortunadamente, existen dos definiciones ampliamente aceptadas de esta función:

$$f_1(x) = \frac{\operatorname{sen}(\pi x)}{\pi x} \quad \text{y} \quad f_2(x) = \frac{\operatorname{sen} x}{x}$$

Estas dos funciones tienen una forma indeterminada de 0/0 cuando $x = 0$. En este caso, se usó el teorema de l'Hôpital del cálculo para probar que ambas funciones son iguales a 1 cuando x = cero. Para valores de x distintos a cero, las dos funciones tienen una forma similar. La primera función, $f_1(x)$, cruza el eje x cuando x es un entero; la segunda función cruza el eje x cuando x es un múltiplo de π.

Suponga que a usted le gustaría definir una función llamada **sinc_x** que usa la segunda definición. Pruebe su función al calcular valores de **sinc_x** para **x** desde -5π hasta $+5\pi$ y grafique los resultados.

1. Establezca el problema.
   Crear y probar una función llamada **sinc_x**, a partir de la segunda definición:

$$f_2(x) = \frac{\operatorname{sen} x}{x}$$

2. Describa las entradas y salidas.

   Entrada Sea x que varía desde -5π hasta 5π

   Salida Crear una gráfica de **sinc_z** contra **x**

3. Desarrolle un ejemplo a mano.
4. Desarrolle una solución MATLAB.
   Destaque su función en un diagrama de flujo como se muestra en la figura 8.7. Luego convierta el diagrama de flujo a comentarios seudocódigo e inserte el código MATLAB apropiado.

Tabla 8.4 Cálculo de la función sinc

x	sen(x)	sinc_x(x) = sen(x)/x
0	0	0/0 = 1
$\pi/2$	1	$1/(\pi/2) = 0.637$
π	0	0
$-\pi/2$	-1	$-1/(\pi/2) = -0.637$

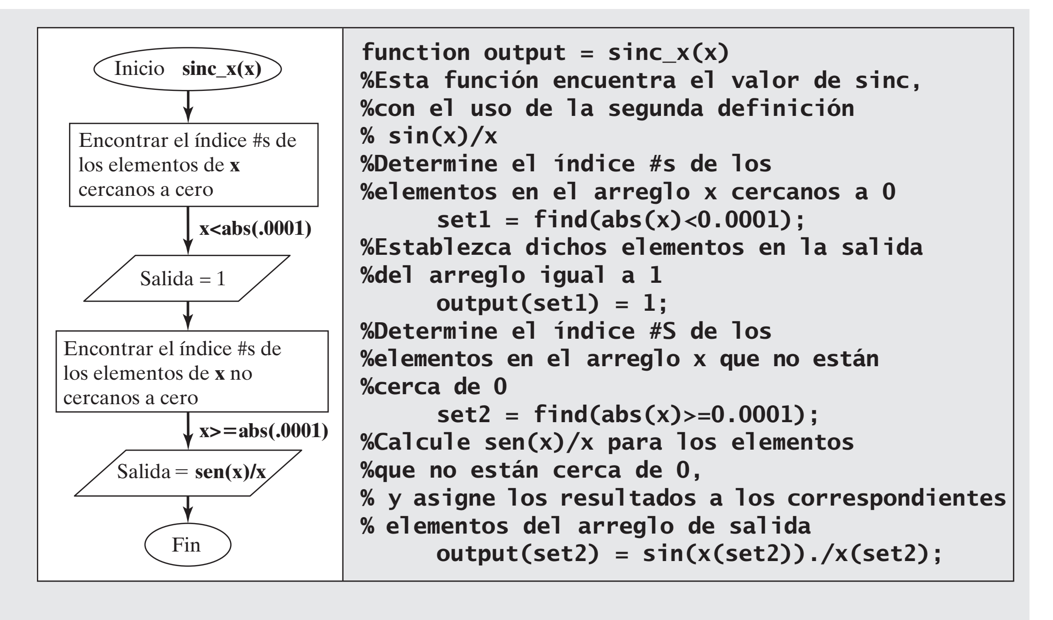


Figura 8.7
Diagrama de flujo de la función sinc.

Una vez que se crea la función se deberá probar en la ventana de comandos:

```
sinc_x(0)
ans =
      1
sinc_x(pi/2)
ans =
      0.6366
sinc_x(pi)
ans =
      3.8982e-017
sinc_x(-pi/2)
ans =
      0.6366
```

Note que **sinc_x(pi/2)** es igual a un número muy pequeño, pero no cero. Por eso MATLAB trata a π como un número de punto flotante y usa una aproximación de su valor real.

5. Ponga a prueba la solución.
   Cuando se comparan los resultados con los del ejemplo a mano, se ve que las respuestas concuerdan. Ahora se puede usar la función con confianza en el problema. Se tiene

```
%Ejemplo 8.1
      clear, clc
%Defina un arreglo de ángulos
      x=-5*pi:pi/100:5*pi;
%Calcule sinc_x
      y=sinc_x(x);
%Cree la gráfica
      plot(x,y)
      title('Función sinc'), xlabel('ángulo,
      radianes'),ylabel('sinc')
```

que genera la gráfica de la figura 8.8.

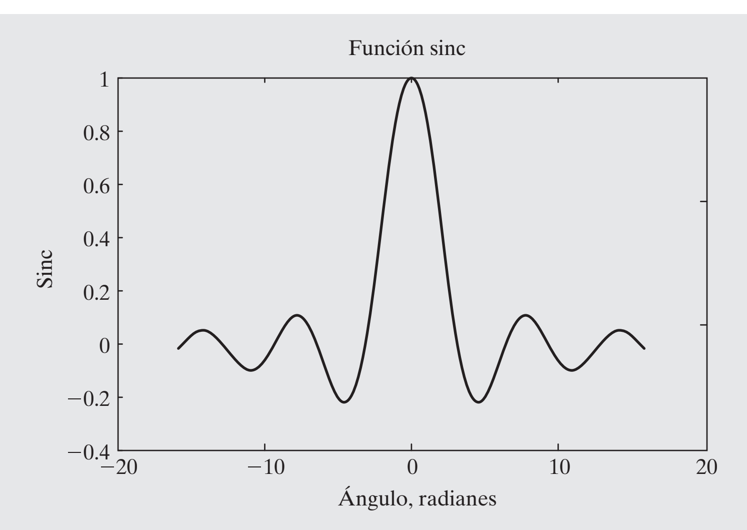


Figura 8.8
La función sinc.

La gráfica también soporta la creencia de que la función opera de manera adecuada. Poner a prueba **sinc_x** con un valor a la vez valida sus respuestas para una entrada escalar; sin embargo, el programa que generó la gráfica envía un argumento vectorial a la función. La gráfica confirma que también se desempeña de manera adecuada con entrada vectorial.

Si tiene problemas para entender cómo funciona esta función, remueva los puntos y coma que suprimen la salida y corra el programa. Entender la salida de cada línea le ayudará a entender mejor la lógica del programa.

Además de **find**, MATLAB ofrece otras dos funciones lógicas: **all** y **any**. La función **all** verifica si una condición lógica es verdadera para *todo* miembro de un arreglo, y la función **any** verifica si una condición lógica es verdadera para *algún* miembro de un arreglo. Consulte la función interna **help** de MATLAB para más información.

Ejercicio de práctica 8.1

Considere las siguientes matrices:

$$x = \begin{bmatrix} 1 & 10 & 42 & 6 \\ 5 & 8 & 78 & 23 \\ 56 & 45 & 9 & 13 \\ 23 & 22 & 8 & 9 \end{bmatrix} \quad y = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 10 & 12 \\ 7 & 21 & 27 \end{bmatrix} \quad z = [10 \quad 22 \quad 5 \quad 13]$$

1. Con la notación de índice solo, encuentre los números índice de los elementos en cada matriz que contengan valores mayores que 10.
2. Encuentre los números de fila y columna (a veces llamados subíndices) de los elementos en cada matriz que contengan valores mayores que 10.
3. Encuentre los valores en cada matriz que sean mayores que 10.
4. Con la notación de índice solo, encuentre los números índice de los elementos en cada matriz que contengan valores mayores que 10 y menores que 40.
5. Encuentre los números de fila y columna para los elementos en cada matriz que contengan valores mayores que 10 y menores que 40.
6. Encuentre los valores en cada matriz que sean mayores que 10 y menores que 40.

7. Con la notación de índice solo, encuentre los números índice de los elementos en cada matriz que contengan valores entres 0 y 10 o entre 70 y 80.
8. Use el comando **length** junto con los resultados del comando **find** para determinar cuántos valores en cada matriz están entre 0 y 10 o entre 70 y 80.

8.4 ESTRUCTURAS DE SELECCIÓN

La mayoría de las veces, el comando **find** puede y debe utilizarse en vez de un enunciado **if**. Sin embargo, hay situaciones en las cuales se requiere el enunciado **if**. Esta sección describe la sintaxis que se emplea en los enunciados **if**.

8.4.1 El if simple

Un enunciado **if** simple tiene la siguiente forma:

```
if enunciados de
   comparación
end
```

Si la comparación (una expresión lógica) es verdadera, se ejecutan los enunciados entre el enunciado **if** y el enunciado **end**. Si la comparación es falsa, el programa salta inmediatamente al enunciado que sigue a **end**. Una buena práctica de programación es sangrar los enunciados dentro de una estructura **if** para legibilidad. Sin embargo, recuerde que MATLAB ignora el espacio en blanco. Sus programas correrán sin importar si usted sangró o no alguna de sus líneas de código.

He aquí un ejemplo realmente simple de un enunciado **if**:

```
if G<50
   count = count +1;
   disp(G);
end
```

Idea clave: los enunciados if, por lo general, funcionan mejor con escalares.

Este enunciado (desde **if** hasta **end**) es fácil de interpretar si **G** es un escalar. Si **G** es menor que 50, entonces se ejecutan los enunciados entre las líneas **if** y **end**. Por ejemplo, si **G** tiene un valor de 25, entonces **count** aumenta por 1 y **G** se despliega en la pantalla. Sin embargo, si **G** no es escalar, entonces el enunciado **if** considera la comparación verdadera **¡sólo si es verdadera para todo elemento!** Por tanto, si G se define desde 0 hasta 80,

```
G = 0:10:80;
```

entonces la comparación es falsa ¡y se ejecutan los enunciados dentro del enunciado **if**! En general, los enunciados **if** funcionan mejor cuando tratan con escalares.

8.4.2 La estructura if/else

El **if** simple le permite ejecutar una serie de enunciados si una condición es verdadera y saltar dichos pasos si la condición es falsa. La cláusula **else** le permite ejecutar un conjunto de enunciados si la comparación es verdadera y un conjunto diferente de enunciados si la comparación es falsa. Suponga que usted quiere sacar el logaritmo de una variable *x*. Usted sabe, por las clases de álgebra básica, que la entrada a la función log debe ser mayor a 0. A continuación hay un conjunto de enunciados **if/else** que calculan el logaritmo si la entrada es positiva y envía un mensaje de error si la entrada a la función es 0 o negativa:

```
if x >0
   y = log(x)
```

```
else
   disp('La entrada a la función log debe ser positiva')
end
```

Cuando **x** es un escalar, esto es fácil de interpretar. Sin embargo, cuando **x** es una matriz, la comparación es verdadera sólo si es verdadera para todo elemento en la matriz. De modo que, si

```
x = 0:0.5:2;
```

entonces los elementos en la matriz no son todos mayores a 0. Por tanto, MATLAB brinca a la porción **else** del enunciado y despliega el mensaje de error. El enunciado **if/else** probablemente está mejor confinado para usar con escalares, aunque puede encontrar que es de uso limitado con vectores.

Sugerencia

MATLAB incluye una función llamada **beep** que hace que la computadora "suene" al usuario. Puede usar esta función para alertar al usuario de un error. Por ejemplo, en la cláusula **if/else**, podría agregar un beep a la porción del código que incluye un enunciado de error:

```
if x >0
   y = log(x)
else
  beep
  disp('La entrada a la función log debe ser
   positiva')
end
```

8.4.3 La estructura elseif

Cuando se anidan varios niveles de enunciados **if/else**, puede ser difícil determinar cuáles expresiones lógicas deben ser verdaderas (o falsas) con la finalidad de ejecutar cada conjunto de enunciados. La función **elseif** le permite comprobar criterios múltiples mientras se mantiene el código fácil de leer. Considere las siguientes líneas de código que evalúan si se emite una licencia de conductor, con base en la edad del solicitante:

```
if age<16
   disp('Lo siento. Tendrá que esperar')
elseif age<18
   disp('Puede obtener un permiso para conducir')
elseif age<70
   disp('Puede obtener una licencia estándar')
else
   disp('Los conductores mayores de 70 requieren una licencia
   especial')
end
```

En éste, MATLAB primero verifica si **age<16**. Si la comparación es verdadera, el programa ejecuta la siguiente línea o conjunto de líneas, despliega el mensaje **Lo siento. Tendrá que esperar**, y luego sale de la estructura **if**. Si la comparación es falsa, MATLAB se mueve a la siguiente comparación **elseif** y esta vez verifica si **age<18**. El programa continúa a través de la estructura **if** hasta que finalmente encuentra una comparación verdadera o hasta que encuentra **else**. Note que la línea **else** no incluye una comparación, pues se ejecuta si el **elseif** inmediatamente anterior es falso.

El diagrama de flujo para esta secuencia de comandos (figura 8.9) usa la forma diamante para indicar una estructura de selección.

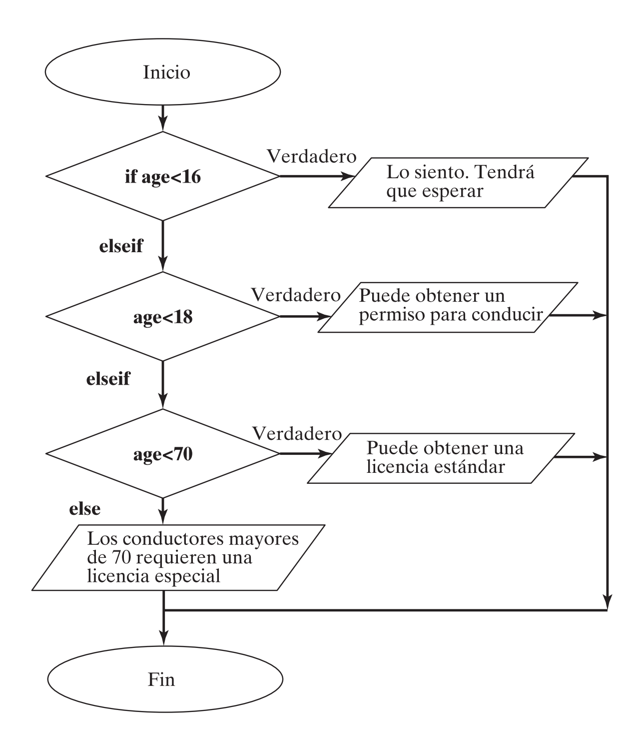


Figura 8.9
Diagrama de flujo que usa enunciados **if** múltiples.

Esta estructura es fácil de interpretar si **age** es un escalar. Si es una matriz, la comparación debe ser verdadera para todo elemento en la matriz. Considere esta matriz de edad

```
age = [15,17,25,55,75]
```

La primera comparación, **is age<16**, es falsa, porque no es verdad para todo elemento en el arreglo. La segunda comparación, **elseif age<18**, también es falsa. La tercera comparación, **elseif age<70**, es falsa también, pues no todas las edades están por abajo de 70. El resultado es **Los conductores mayores de 70 requieren una licencia especial**, un resultado que no les gustará a los otros conductores.

Sugerencia

Un error común que cometen los nuevos programadores cuando usan enunciados **if** es especificar en exceso los criterios. En el ejemplo anterior es suficiente establecer que **age < 18** en la segunda cláusula **if**, porque age no puede ser menor que 16 y además satisfacer este enunciado. No necesita especificar **age < 18** y **age >= 16**. Si especifica en exceso los criterios, se arriesga a definir una ruta de cálculo para la que no hay respuesta correcta. Por ejemplo, en el código

```
if age<16
   disp('Lo siento. Tendrá que esperar')
elseif age<18 & age>16
   disp('Puede obtener un permiso para conducir')
elseif age<70 & age>18
   disp('Puede obtener una licencia estándar')
elseif age>70
   disp('Los conductores mayores de 70 requieren
     una licencia especial')
end
```

no hay opción correcta para age = 16, 18 o 70.

En general, las estructuras **elseif** funcionan bien para escalares, pero probablemente **find** es mejor opción para matrices. He aquí un ejemplo que usa **find** con un arreglo de edades y genera una tabla de resultados en cada categoría:

```
age = [15,17,25,55,75];
set1 = find(age<16);
set2 = find(age>=16 & age<18);
set3 = find(age>=18 & age<70);
set4 = find(age>=70);

fprintf('Lo siento. Tendrá que esperar %3.0f
  \n',age(set1))
fprintf('Puede obtener un permiso para conducir %3.0f
  \n',age(set2))
fprintf('Puede obtener una licencia estándar
  %3.0f \n',age(set3))
fprintf('Los conductores mayores de 70 requieren
  una licencia especial %3.0f \n',age(set4))
```

Estos comandos regresan

```
Lo siento. Tendrá que esperar. Sólo tiene 15
Puede obtener un permiso para conducir porque tiene 17
Puede obtener una licencia estándar porque tiene 25
Puede obtener una licencia estándar porque tiene 55
Los conductores mayores de 70 requieren una licencia
  especial. Usted tiene 75
```

Dado que en esta secuencia se evalúa cada **find**, es necesario especificar el rango completamente (por ejemplo, **age>=16 & age<18**).

EJEMPLO 8.2

Asignación de calificaciones

La familia de enunciados **if** se usa de manera más efectiva cuando la entrada es un escalar. Cree una función para determinar calificaciones de examen con base en los puntos y suponga una sola entrada a la función. Las calificaciones se deben basar en los siguientes criterios:

Calificación	Promedio
A	90 a 100
B	80 a 90
C	70 a 80
D	60 a 70
E	<60

1. Establezca el problema.
   Determinar la calificación obtenida en un examen.

2. Describa las entradas y salidas.

   Entrada Puntaje único, no un arreglo

   Salida Calificación en letra

3. Desarrolle un ejemplo a mano.
   85 debería ser una B
   Pero, ¿90 debería ser una A o una B? Es necesario crear criterios más exactos.

Calificación	Promedio
A	≥90 a 100
B	≥80 y <90
C	≥70 y <80
D	≥60 y <70
E	<60

4. Desarrolle una solución MATLAB.
   Resalte la función con el diagrama de flujo que se muestra en la figura 8.10.

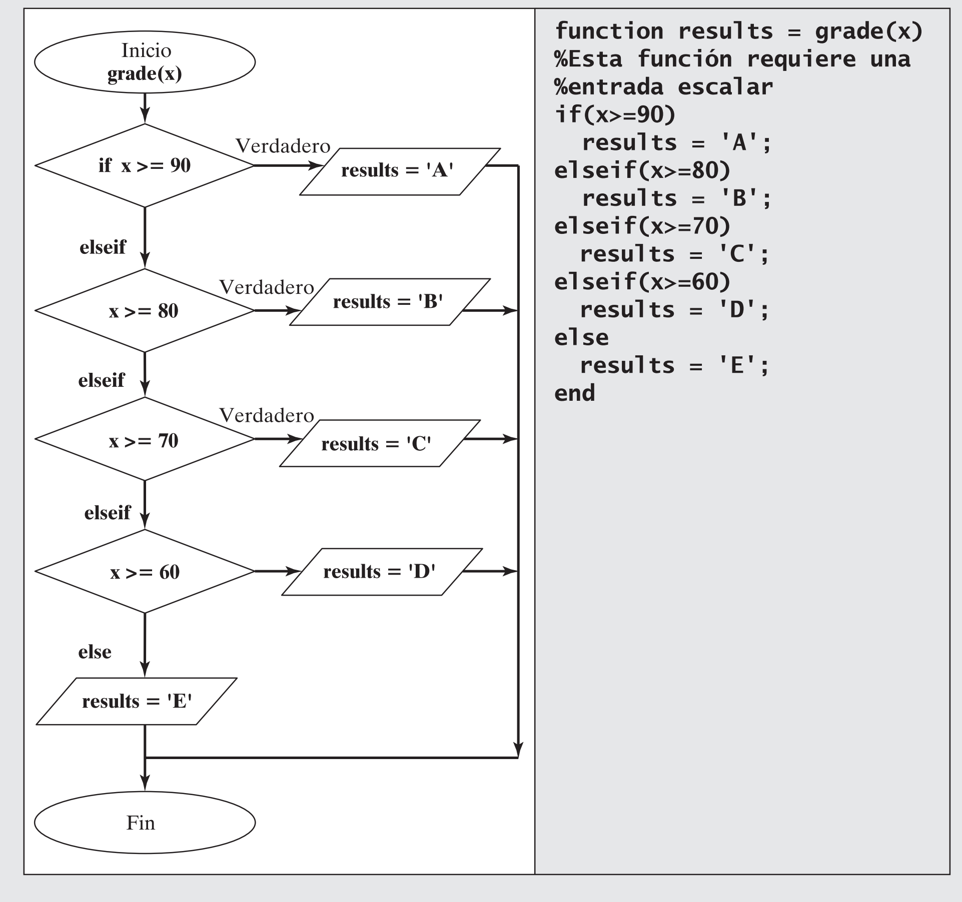


Figura 8.10
Diagrama de flujo para un esquema de calificación.

5. Ponga a prueba la solución.
   Ahora pruebe la función en la ventana de comandos:

```
grade(25)
ans =
E
grade(80)
ans =
B
grade(-52)
ans =
E
grade(108)
ans =
A
```

Note que aunque la función parece funcionar de manera adecuada regresa calificaciones para valores arriba de 100 y valores menores que 0. Si le parece, ahora puede regresar y agregar la lógica para excluir dichos valores:

```
function results = grade(x)
%Esta función requiere una entrada escalar
if(x>=0 & x<=100)
if(x>=90)
   results = 'A';
elseif(x>=80)
   results = 'B';
elseif(x>=70)
   results = 'C';
elseif(x>=60)
   results = 'D';
else
   results = 'E';
end
else
   results = 'Entrada ilegal';
end
```

Se puede probar de nuevo la función en la ventana de comandos:

```
grade(-10)
ans =
Entrada ilegal
grade(108)
ans =
Entrada ilegal
```

Esta función operará mejor para escalares, pero si envía un vector a la función, puede obtener algunos resultados inesperados, como

```
score = [95,42,83,77];
grade(score)
ans =
E
```

Ejercicio de práctica 8.2

La familia de funciones **if** es particularmente útil en funciones. Escriba una función para cada uno de estos problemas si supone que la entrada a la función es un escalar:

1. Suponga que en un estado de Estados Unidos la edad legal para beber es 21. Escriba y pruebe una función para determinar si una persona es lo suficientemente madura para beber.
2. Muchos juegos en los parques de diversiones requieren que los usuarios tengan cierta estatura mínima. Suponga que la estatura mínima es 48″ para cierto juego. Escriba y pruebe una función para determinar si el usuario es lo suficientemente alto.
3. Cuando una parte se fabrica, las dimensiones usualmente se especifican con una tolerancia. Suponga que cierta parte necesita tener 5.4 cm de largo, más o menos 0.1 cm (5.4 $\pm$ 0.1 cm). Escriba una función para determinar si una parte está dentro de dichas especificaciones.
4. Desafortunadamente, Estados Unidos actualmente usa unidades tanto métricas como inglesas. Suponga que la parte del problema 3 se inspeccionó al medir la longitud en pulgadas en lugar de cm. Escriba y pruebe una función que determine si la parte está dentro de las especificaciones y que acepte entrada en la función en pulgadas.
5. Muchos motores de cohete sólido constan de tres etapas. Una vez que la primera etapa se quema, se separa del misil y la segunda etapa se enciende. Luego la segunda etapa se quema y separa, y la tercera etapa se enciende. Finalmente, una vez que la tercera etapa se quema, también se separa del misil. Suponga que los siguientes datos representan aproximadamente los tiempos durante los que cada etapa se quema:

Etapa 1	0–100 segundos
Etapa 2	100–170 segundos
Etapa 3	170–260 segundos

   Escriba y pruebe una función para determinar si el misil está en etapa de vuelo 1, etapa de vuelo 2, etapa de vuelo 3 o vuelo libre (sin potencia).

8.4.4 Switch y case

La estructura **switch/case** se usa con frecuencia cuando existe una serie de opciones de ruta de programación para una variable dada, dependiendo de su valor. **Switch/case** es similar a **if/else/elseif**. De hecho, cualquier cosa que pueda hacer con **switch/case** se podría hacer con **if/else/elseif**. Sin embargo, el código es un poco más fácil de leer con **switch/case**, una estructura que le permite elegir entre múltiples salidas, con base en ciertos criterios. Ésta es una importante distinción entre **switch/case** y **elseif**. Los criterios pueden ser un escalar (un

número) o una cadena. En la práctica se usa más con cadenas que con números. La estructura de **switch/case** es

```
switch  variable
case  option1
  código a ejecutar si la variable es igual a la opción 1
case  option2
  código a ejecutar si la variable es igual a la opción 2
                                  ⋮
case  option_n
  código a ejecutar si la variable es igual a la opción n
otherwise
  código a ejecutar si la variable no es igual a cualquiera
    de las opciones
end
```

He aquí un ejemplo: suponga que quiere crear una función que diga al usuario cuál es la tarifa aérea a una de tres diferentes ciudades:

```
city = input('Ingrese el nombre de una ciudad en apóstrofes ')
switch city
   case 'Boston'
      disp('$345')
   case 'Denver'
      disp('$150')
   case 'Honolulu'
      disp('Quédese en casa y estudie')
   otherwise
      disp('No en archivo')
end
```

Si al correr este script replica '**Boston**' en el prompt, MATLAB responde

```
city =
Boston
$345
```

Puede decir al comando **input** que espere una cadena al agregar 's' en un segundo campo. Esto libera al usuario del abrumador requisito de agregar apóstrofes alrededor de cualquier cadena de entrada. Con la 's' agregada, el código precedente ahora se lee como sigue:

```
city = input('Ingrese el nombre de una ciudad en apóstrofes ','s')
switch city
   case 'Boston'
     disp('$345')
   case 'Denver'
     disp('$150')
   case 'Honolulu'
     disp('Quédese en casa y estudie')
   otherwise
     disp('No en archivo')
end
```

La porción **otherwise** de la estructura **switch/case** no se requiere para que funcione la estructura. Sin embargo, debe incluirla si hay alguna forma de que el usuario pueda ingresar un valor no igual a uno de los casos.

Las estructuras **case/switch** tienen diagramas de flujo exactamente iguales a las estructuras **if/else**.

Sugerencia

Si usted es programador C, puede haber usado **switch/case** en dicho lenguaje. Una diferencia importante en MATLAB es que, una vez que se encuentra un caso "verdadero", el programa no verifica los otros casos.

EJEMPLO 8.3

Compra de gasolina

Existen cuatro países en el mundo que oficialmente no usan el sistema métrico: Estados Unidos, Reino Unido, Liberia y Myanmar. Incluso, en Estados Unidos la práctica es que algunas industrias sean casi completamente métricas y otras todavía usen el sistema inglés de unidades. Por ejemplo, cualquier mecánico le dirá que, aunque los automóviles antiguos tengan una mezcla de componentes, algunos métricos y otros ingleses, los automóviles nuevos (cualquier automóvil construido después de 1989) casi son completamente métricos. El vino se envasa en litros, pero la leche se envasa en galones. Los estadounidenses miden distancia en millas, pero la electricidad en watts. La confusión entre unidades métricas e inglesas es común. Los estadounidenses que viajan a Canadá, por lo general, se confunden debido a que la gasolina se vende en litros en Canadá, pero en Estados Unidos en galones.

Imagine que usted quiere comprar gasolina (figura 8.11). escriba un programa que

- pregunte al usuario si quiere ordenar la gasolina en litros o en galones.
- conmine al usuario a ingresar cuántas unidades quiere comprar.
- calcule el costo total al usuario, si supone que la gasolina cuesta $2.89 por galón.

Use una estructura **switch/case**.

1. Establezca el problema.
   Calcular el costo de una compra de gasolina.



Figura 8.11
La gasolina se vende tanto en litros como en galones.

2. Describa las entradas y salidas.

   Entrada Especificar galones o litros
   Número de galones o litros

   Salida Costo en dólares, si supone \$2.89 por galón

3. Desarrolle un ejemplo a mano.
   Si el volumen se especifica en galones, el costo es

$$\text{volumen} \times \$2.89$$

   de modo que, para 10 galones,

$$\text{costo} = 10 \text{ galones} \times \$2.89/\text{galón} = \$28.90$$

   Si el volumen se especifica en litros, se necesita convertir litros a galones y luego calcular el costo:

$$\text{volumen} = \text{litros} \times 0.264 \text{ galón/litro}$$

$$\text{costo} = \text{volumen} \times \$2.89$$

   De modo que, para 10 litros,

$$\text{volumen} = 10 \text{ litros} \times 0.264 \text{ galón/litro} = 2.64 \text{ galones}$$
$$\text{costo} = 2.64 \text{ galones} \times 2.89 = \$7.63$$

4. Desarrolle una solución MATLAB.
   Primero cree un diagrama de flujo (figura 8.12). Luego convierta el diagrama de flujo en comentarios seudocódigo. Finalmente, agregue el código MATLAB:

```
 clear,clc
%Defina el costo por galón
 rate = 2.89;
%Pida al usuario ingresar galones o litros
 unit = input('Ingrese galones o litros\n ','s');
%Use un switch/case para determinar el factor de conversión
 switch unit
  case 'galones'
    factor = 1;
  case 'litros'
    factor = 0.264;
  otherwise
    disp('No disponible')
    factor = 0;
 end
%Pregunte al usuario cuánta gasolina quiere comprar
 volume = input( ['Ingrese el volumen que le gustaría comprar
   en ',unit,': \n'] );
%Calcula el costo de la gasolina
if factor ~=0
  cost = volume * factor*rate;
%Envía los resultados a la pantalla
 fprintf('Serán $ %5.2f por %5.1f %s
         \n',cost,volume,unit)
end
```

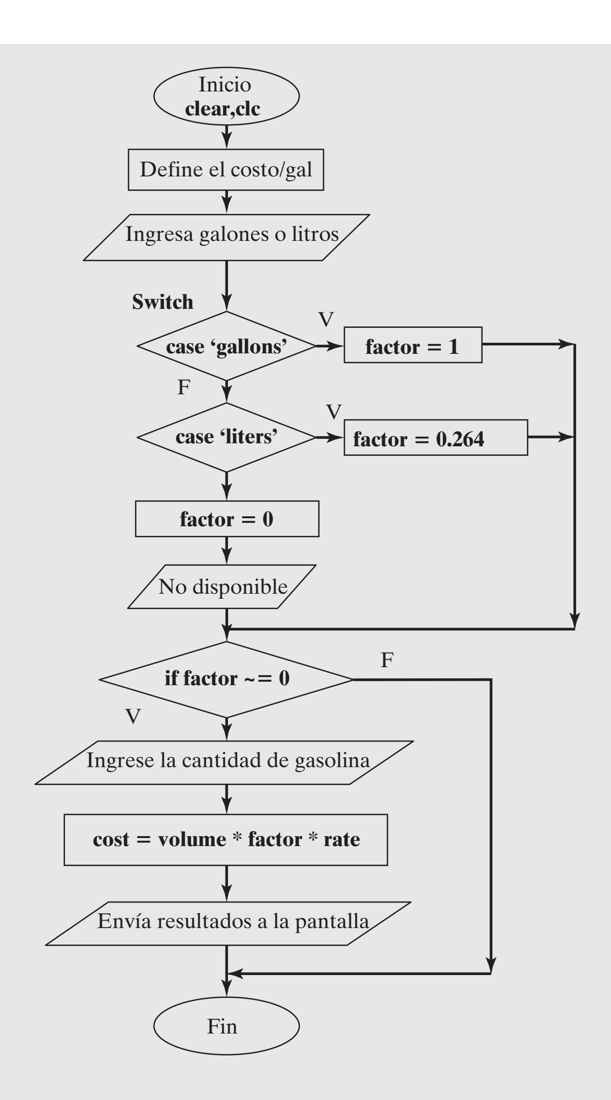


Figura 8.12
Diagrama de flujo para determinar el costo de gasolina con la estructura **switch/case**.

Hay varias cosas que observar en esta solución. Primero, la variable **unit** contiene un arreglo de información carácter. Si verifica la ventana del área de trabajo después de correr este programa, notará que **unit** es un arreglo carácter 1×6 (si ingresó litros) o un arreglo carácter 1×7 (si ingresó galones).

En la línea

```
unit = input('Ingrese galones o litros ','s');
```

el segundo campo, '**s**', le dice a MATLAB que espere una cadena como entrada. Esto permite al usuario ingresar galones o litros sin los apóstrofes.

En la línea

```
volume = input( ['Ingrese el volumen que le gustaría comprar
                en ',unit,': '] );
```

se creó un arreglo carácter a partir de tres componentes:

- la cadena '**Ingrese el volumen que le gustaría comprar en**'.
- la variable carácter **unit**.
- la cadena '**:**'.

Al combinar estos tres componentes, fue capaz de hacer que el programa propusiera al usuario

```
Ingrese el volumen que le gustaría comprar en litros
```

o

```
Ingrese el volumen que le gustaría comprar en galones
```

En el enunciado **fprintf**, se incluyó un campo para entrada de cadena al usar el marcador de posición **%s**:

```
fprintf('Serán $ %5.2f por %5.1f %s
        \n',cost,volume,unit)
```

Esto permitió al programa pedir a los usuarios que la gasolina se midiera o en galones o en litros.

Finalmente, se usó un enunciado **if**, de modo que si el usuario ingresó algo además de galones o litros, no se realizaran cálculos.

5. Ponga a prueba la solución.
Se puede probar la solución corriendo el programa tres veces separadas, una vez para galones, una vez para litros y otra para alguna unidad no soportada. La interacción en la ventana de comandos para galones es

```
Ingrese galones o litros
galones
Ingrese el volumen que le gustaría comprar en galones:
10
Serán $ 28.90 por 10.0 galones
```

Para litros, la interacción es

```
Ingrese galones o litros
litros
Ingrese el volumen que le gustaría comprar en litros:
10
Serán $ 7.63 por 10.0 litros
```

Finalmente, si usted ingresa algo aparte de galones o litros, el programa envía un mensaje de error a la ventana de comando:

```
Ingrese galones o litros
cuartos
No disponible
```

Dado que los resultados del programa son los mismos que el cálculo a mano, parece que el programa funciona como se planeó.

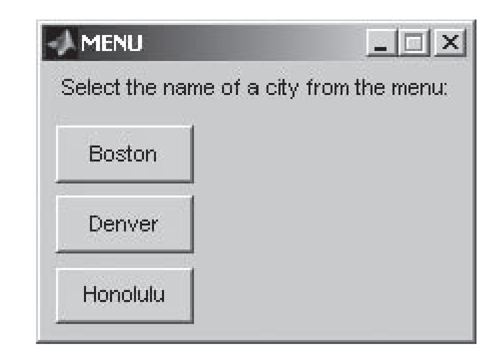


Figura 8.13
Ventana de menú desplegable.

8.4.5 Menu

Idea clave: las interfaces de usuario gráficas, como el recuadro de menú, reducen la posibilidad de que el usuario cometa errores, como equivocaciones de deletreo.

La función **menu** se usa con frecuencia en conjunto con una estructura **switch/case**. Esta función hace que aparezca un recuadro de menú en la pantalla, con una serie de botones definidos por el programador

```
input = menu('Mensaje al usuario','texto para botón
  1','texto para botón 2', etc.)
```

Se puede usar la opción **menu** en el ejemplo de tarifa aérea anterior para asegurar que el usuario elija sólo ciudades de las que se tiene información. Esto también significa que no se necesita la sintaxis **otherwise**, pues no es posible elegir una ciudad que "no esté en archivo".

```
city = menu('Select the name of a city from the menu:
  ','Boston','Denver','Honolulu')
switch city
    case 1
        disp('$345')
    case 2
        disp('$150')
    case 3
        disp('Quédese en casa y estudie')
end
```

Note que un caso número sustituyó la cadena en cada línea **case**. Cuando se ejecuta el script, aparece el recuadro menú que se muestra en la figura 8.13 y espera que el usuario seleccione uno de los botones. Si eligió Honolulu, MATLAB responderá

```
city =
    3
Quédese en casa y estudie
```

Desde luego, podría suprimir la salida del comando **disp**, que se incluyó aquí por claridad.

EJEMPLO 8.4

Compra de gasolina: un enfoque de menu

En el ejemplo 8.3 se usó un enfoque **switch/case** para determinar si el cliente quiere comprar gasolina medida en galones o en litros. Un problema con el programa es que, si el usuario no

puede deletrear, el programa no funcionará. Por ejemplo, si cuando se piden galones o litros el usuario ingresa

```
litters
```

El programa responderá

```
No disponible
```

Este problema se puede resolver al usar un menú; entonces el usuario sólo necesita presionar un botón para hacer una elección. Todavía se usará la estructura **switch/case**, pero se combinará con el menú.

1. Establezca el problema.
   Calcular el costo de una compra de gasolina.
2. Describa las entradas y salidas.

   Entrada Especificar galones o litros desde un menú
   Número de galones o litros

   Salida Costo en dólares, si supone $2.89 por galón

3. Desarrolle un ejemplo a mano.
   Si el volumen se especifica en galones, el costo es

$$\text{volumen} \times \$2.89$$

   De modo que para 10 galones,

$$\text{costo} = 10 \text{ galones} \times \$2.89/\text{galón} = \$28.90$$

   Si el volumen se especifica en litros, se necesita convertir litros a galones y luego calcular el costo:

$$\text{volumen} = \text{litros} \times 0.264 \text{ galón/litro}$$
$$\text{costo} = \text{volumen} \times \$2.89$$

   De modo que para 10 litros,

$$\text{volumen} = 10 \text{ litros} \times 0.264 \text{ galón/litro} = 2.64 \text{ galones}$$
$$\text{costo} = 2.64 \text{ galones} \times 2.89 = \$7.63$$

4. Desarrolle una solución MATLAB.
   Primero cree un diagrama de flujo (figura 8.14). Luego convierta el diagrama de flujo en comentarios de seudocódigo. Finalmente, agregue el código MATLAB:

```
%Ejemplo 8.4
clear,clc
%Defina el costo por galón
 rate = 2.89;
%Pida al usuario ingresar galones o litros desde un menú
 disp('Use el menú para hacer su selección')
 choice = menu('¿Medir gasolina en litros o galones?',
'galones','litros');
%Use switch/case para determinar el factor de conversión
switch choice
  case 1
    factor = 1;
    unit = 'galones'
```

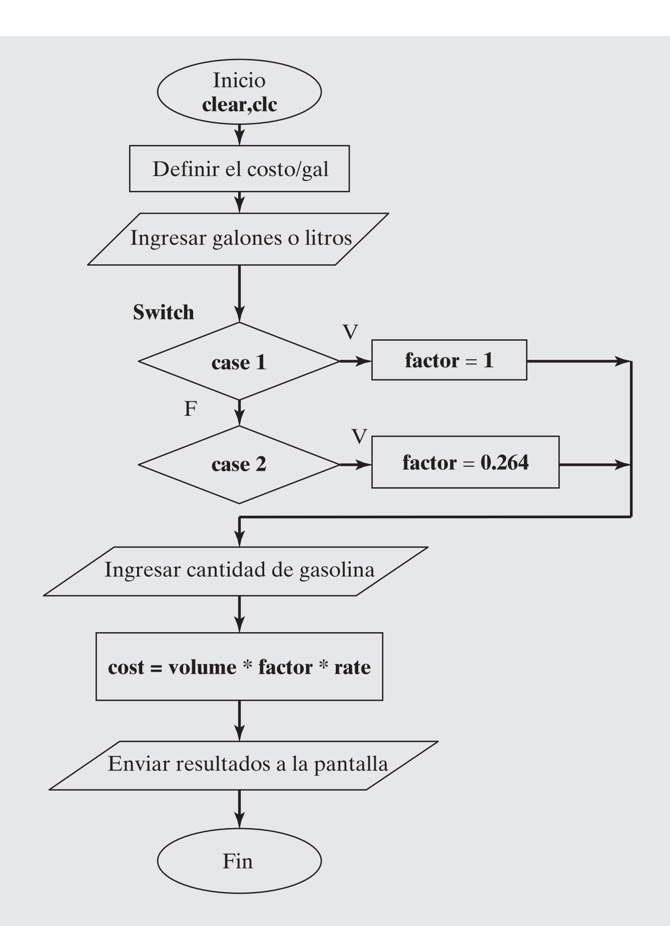


Figura 8.14
Diagrama de flujo para determinar el costo de gasolina desde un menú.

```
  case 2
    factor = 0.264;
    unit = 'litros'
end
%Pregunte al usuario cuánta gasolina le gustaría comprar
 volume = input( ['Ingrese el volumen que le gustaría
   comprar en ',unit,': \n'] );
%Calcule el costo de la gasolina
 cost = volume * factor*rate;
%Envíe los resultados a la pantalla
 fprintf('Serán $ %5.2f por %5.1f %s
   \n',cost,volume,unit)
```

Esta solución es más simple que la del ejemplo 8.3 porque no hay oportunidad de una mala entrada. Sin embargo, hay que señalar algunas cosas.

Cuando se define la opción mediante la función menu, el resultado es un número, no un arreglo carácter:

```
choice = menu('¿Medir gasolina en litros o
  galones?', 'galones','litros');
```

Usted puede comprobar esto al consultar la ventana del área de trabajo, en la que la opción se menciona como un número 1 $\times$ 1 de doble precisión.

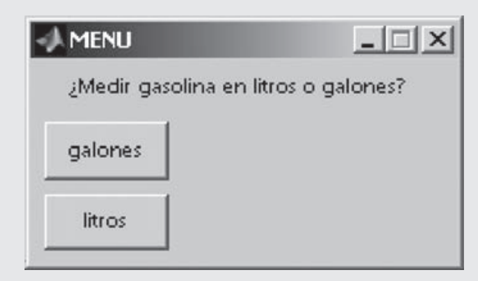
Puesto que no se usó el comando **input** para definir la variable **unit**, que es una cadena (un arreglo carácter), se necesita especificar el valor de **unit** como parte de los cálculos de caso:

```
case 1
  factor = 1;
  unit = 'galones'
case 2
  factor = 0.264;
  unit = 'litros'
```

Hacer esto le permite usar el valor de **unit** en la salida de la ventana de comandos, tanto en el comando **disp** como en **fprintf**.

5. Ponga a prueba la solución.
Como en el ejemplo 8.3, se puede probar la solución al correr el programa, pero esta vez se necesita intentarlo sólo dos veces: una vez para galones y una para litros. La interacción en la ventana de comandos para galones es

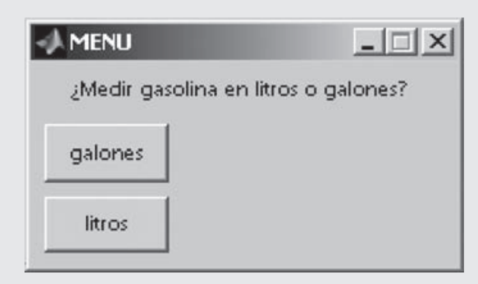
```
Use el menú para hacer su selección
```




```
Ingrese el volumen que le gustaría comprar en galones:
10
Será $28.90 por 10.0 galones
```

Para litros, la interacción es

```
Use el menú para hacer su selección
```




```
Ingrese el volumen que le gustaría comprar en litros:
10
Serán $7.63 por 10.0 litros
```

Estos valores concuerdan con los del ejemplo desarrollado a mano y tienen la ventaja añadida de que no es posible deletrear mal alguna entrada.

Ejercicio de práctica 8.3

Use la estructura **switch/case** para resolver estos problemas:

1. Cree un programa que conmine al usuario a ingresar su año en la escuela: primero, segundo, tercero o cuarto. La entrada será una cadena. Use la estructura **switch/case** para determinar qué día serán los finales para cada grupo: lunes para primero, martes para segundo, miércoles para tercero y jueves para cuarto.
2. Repita el problema 1 pero esta vez con un menú.
3. Cree un programa que conmine al usuario a ingresar el número de dulces que le gustaría comprar. La entrada será un número. Use la estructura **switch/case** para determinar la cuenta, donde

1 dulce = \$0.75
2 dulces = \$1.25
3 dulces = \$1.65

más de 3 dulces = \$1.65 + \$0.30(número ordenado – 3)

8.5 ESTRUCTURAS DE REPETICIÓN: BUCLES

Los bucles (loops) se usan cuando necesita repetir un conjunto de instrucciones muchas veces. MATLAB soporta dos tipos diferentes de bucles: el bucle **for** y el bucle **while**. Los bucles **for** son la opción más sencilla cuando usted sabe cuántas veces necesita repetir el bucle. Los bucles **while** son las opciones más sencillas cuando necesita mantener la repetición de las instrucciones hasta que se satisface un criterio. Si tiene experiencia de programación previa, puede estar tentado a usar bucles de manera extensa. Sin embargo, se pueden componer programas MATLAB que eviten los bucles, ya sea mediante el comando **find** o mediante la vectorización del código. (En la vectorización se opera sobre vectores enteros a la vez, en lugar de un elemento a la vez.) Es buena idea evitar los bucles siempre que sea posible, porque los programas resultantes corren más rápido y con frecuencia requieren menos pasos de programación.

8.5.1 Bucles for

La estructura del bucle **for** es simple. La primera línea identifica el bucle y define un índice, que es un número que cambia en cada paso a través del bucle. Después de la línea de identificación viene el grupo de comandos que se quiere ejecutar. Finalmente, la terminación del bucle se identifica mediante el comando **end**. En resumen, se tiene

```
for index = [matrix]
      comandos a ejecutar
end
```

El bucle se ejecuta una vez para cada elemento de la matriz índice identificada en la primera línea. He aquí un ejemplo realmente simple:

```
for k=[1,3,7]
  k
end
```

Este código regresa

```
k =
    1
k =
    3
```

```
k =
    7
```

En este caso, el índice es **k**. Los programadores usan con frecuencia **k** como una variable índice por cuestiones de estilo. La matriz índice también se puede definir con el operador dos puntos o, de hecho, también en algunas otras formas. He aquí un ejemplo de código que encuentra el valor de 5 elevado a potencias entre 1 y 3:

Idea clave: los bucles le permiten repetir secuencias de comandos hasta que se satisfacen ciertos criterios.

```
for k=1:3
  a=5^k
end
```

En la primera línea, el índice, **k**, se define como la matriz [1,2,3]. La primera vez en el bucle, a **k** se le asigna un valor de 1 y se calcula 5^1. Luego se repite el bucle, pero ahora **k** es igual a 2 y se calcula 5^2. La última vez en el bucle, **k** es igual a 3 y se calcula 5^3. Puesto que los enunciados en el bucle se repiten tres veces, el valor de **a** se despliega tres veces en la ventana de comandos:

```
a =
   5
a =
   25
a =
   125
```

Aunque **k** se definió como una matriz en la primera línea del bucle **for**, dado que **k** es un número índice cuando se usa en el bucle, puede ser igual sólo a un valor a la vez. Después de terminar de ejecutar el bucle, si se llama **k**, sólo tiene un valor: el valor del índice la última vez en el bucle. Para el ejemplo anterior,

```
k
```

regresa

```
k =
   3
```

Note que **k** se menciona como una matriz 1 × 1 en la ventana del área de trabajo.

Una de las formas más comunes de usar un bucle **for** es para definir una nueva matriz. Considere, por ejemplo, el código

```
for k = 1:5
  a(k) = k^2
end
```

Este bucle define una nueva matriz, **a**, un elemento a la vez. Dado que el programa repite su conjunto de instrucciones cinco veces, a la matriz **a** se agrega un nuevo elemento cada vez a lo largo del bucle con la siguiente salida en la ventana de comandos:

```
a =
   1
a =
   1   4
a =
   1   4   9
a =
   1   4   9   16
a =
   1   4   9   16   25
```

Sugerencia

La mayoría de los programas de cómputo no tienen la habilidad de MATLAB para manipular matrices tan fácilmente; por tanto, se apoyan en bucles similares al recién presentado para definir arreglos. Sería más fácil crear el vector **a** en MATLAB con el código

```
k = 1:5
a = k.^2
```

que regresa

```
k =
   1   2   3   4   5
a =
   1   4   9  16  25
```

Éste es un ejemplo de ***vectorización*** del código.

Otro uso común para un bucle **for** es combinarlo con un enunciado **if** y determinar cuántas veces algo es verdadero. Por ejemplo, en la lista de puntajes de examen que se muestra en la primera línea, ¿cuántos están por arriba de 90?

```
scores = [76,45,98,97];
count = 0;
for k=1:length(scores)
   if scores(k)>90
       count = count + 1;
   end
end
disp(count)
```

Cada vez que se activa el bucle, si el puntaje es mayor que 90, el contador aumenta 1.

La mayoría de las veces se crean bucles **for** que usan una matriz índice que es una sola fila. Sin embargo, si en la especificación de índice se define una matriz bidimensional, MATLAB usa toda una columna como el índice cada vez que pasa por el bucle. Por ejemplo, suponga que se define el índice como

$$k = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 9 \\ 1 & 8 & 27 \end{bmatrix}$$

Entonces

```
for k=[1,2,3; 1,4,9; 1,8,27]
   a=k'
end
```

regresa

```
a =
   1    1    1
a =
   2    4    8
a =
   3    9   27
```

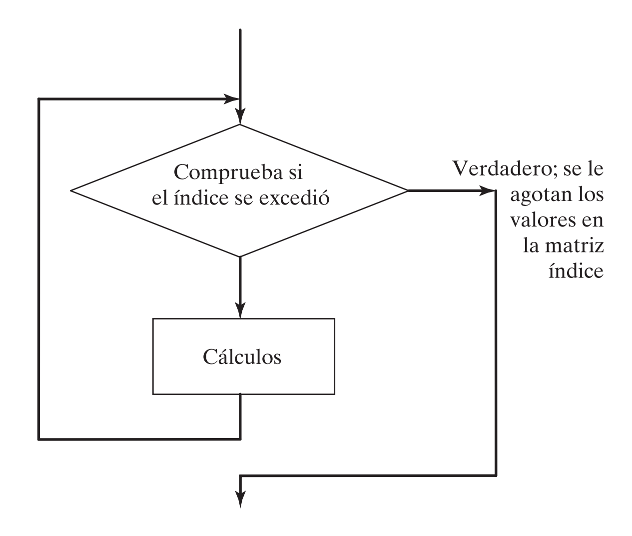


Figura 8.15
Diagrama de flujo para un bucle **for**.

Note que **k** se movió cuando se hizo igual a **a**, de modo que los resultados son filas en lugar de columnas.

Se puede resumir el uso de los bucles for con las siguientes reglas:

Idea clave: use los bucles for cuando sepa cuántas veces necesita repetir una secuencia de comandos.

- El bucle comienza con un enunciado **for** y termina con la palabra **end**.
- La primera línea en el bucle define el número de veces que se repetirán los bucles mediante un número índice.
- El índice de un bucle **for** debe ser una variable. (El índice es el número que cambia cada vez a lo largo del bucle.) Aunque **k** con frecuencia se usa como el símbolo para el índice, se puede emplear cualquier nombre de variable. El uso de **k** es cuestión de estilo.
- Cualquiera de las técnicas aprendidas para definir una matriz se puede usar para definir la matriz índice. Un enfoque común es usar el operador dos puntos, como en

```
for index = start:inc:final
```

- Si la expresión es un vector fila, los elementos se usan uno a la vez una vez cada que se les pasa por el bucle.
- Si la expresión es una matriz (esta alternativa no es común), cada vez que pasa por el bucle el índice contendrá la siguiente *columna* en la matriz. ¡Esto significa que el índice será un vector columna!
- Una vez que completa el bucle **for**, el índice es el último valor utilizado.
- Con frecuencia se evitan los bucles **for** al vectorizar el código.

El diagrama de flujo básico para un bucle **for** incluye un diamante que refleja el hecho de que un bucle **for** comienza cada paso comprobando si hay un nuevo valor en la matriz índice (figura 8.15). Si no lo hay, el bucle termina y el programa continúa con los enunciados después del bucle.

EJEMPLO 8.5

Creación de una tabla grados a radianes

Aunque sería mucho más sencillo usar la capacidad vectorial de MATLAB para crear una tabla grados a radianes, se puede demostrar el uso de los bucles **for** con este ejemplo.

1. Establezca el problema.
   Crear una tabla que convierta valores de ángulo en grados a radianes, desde 0 hasta 360 grados, en incrementos de 10 grados.

2. Describa las entradas y salidas.

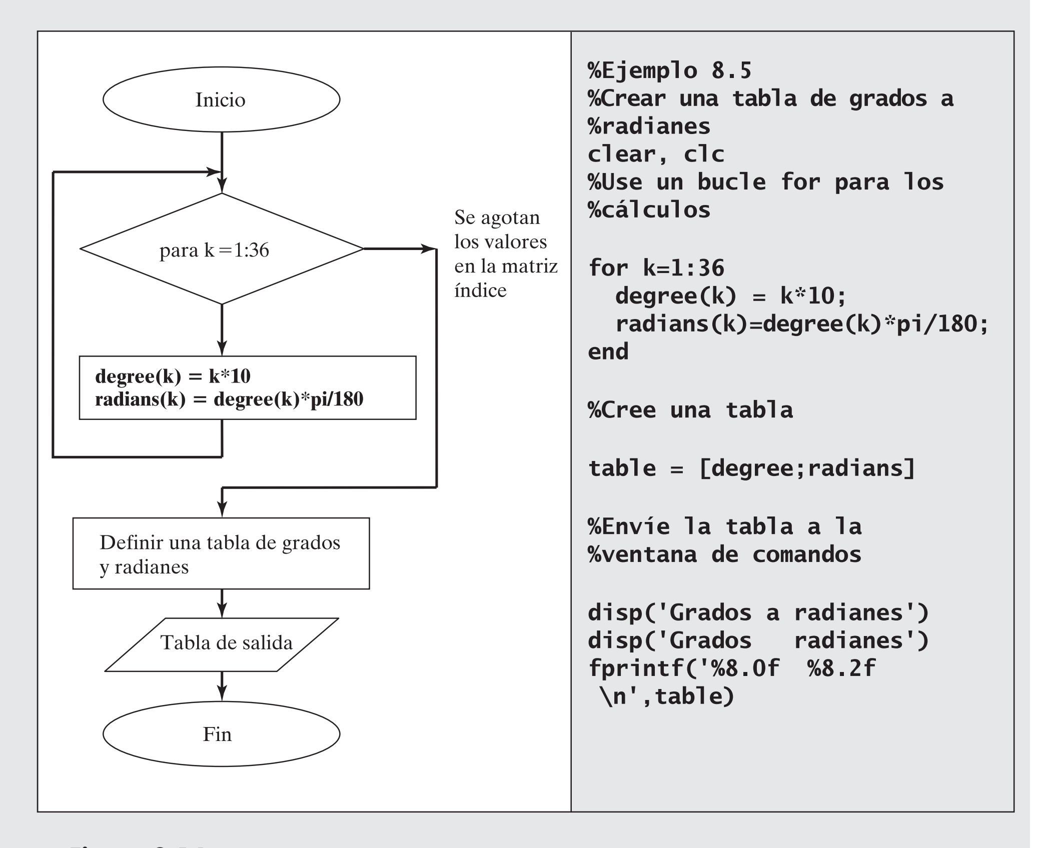
   Entrada Un arreglo de valores de ángulo en grados

   Salida Una tabla de valores de ángulo tanto en grados como en radianes

3. Desarrolle un ejemplo a mano.
   Para 10 grados,

$$\text{radianes} = (10)\frac{\pi}{180} = 0.1745$$

4. Desarrolle una solución MATLAB.
   Desarrolle primero un diagrama de flujo (figura 8.16) para auxiliarse a planear su código.




```
%Ejemplo 8.5
%Crear una tabla de grados a
%radianes
clear, clc
%Use un bucle for para los
%cálculos

for k=1:36
  degree(k) = k*10;
  radians(k)=degree(k)*pi/180;
end

%Cree una tabla

table = [degree;radians]

%Envíe la tabla a la
%ventana de comandos

disp('Grados a radianes')
disp('Grados   radianes')
fprintf('%8.0f  %8.2f
 \n',table)
```

Figura 8.16
Diagrama de flujo para cambiar grados a radianes.

La ventana de comandos despliega los siguientes resultados:

```
Grados a radianes
Grados     Radianes
   10      0.17
   20      0.35
   30      0.52   etc.
```

5. Ponga a prueba la solución.
   El valor para 10 grados calculado por MATLAB es el mismo que el cálculo a mano.
   Obviamente, es mucho más fácil usar las capacidades vectoriales de MATLAB para este cálculo. Usted obtiene exactamente la misma respuesta y toma significativamente menos tiempo de cálculo. Este enfoque se llama vectorización de su código y es una de las fortalezas de MATLAB. El código vectorizado es

```
degrees = 0:10:360;
radians = degrees * pi/180;
table = [degree;radians]
disp('Grados a radianes')
disp('Grados   Radianes')
fprintf('%8.0f  %8.2f \n',table)
```

EJEMPLO 8.6

Cálculo de factoriales con un bucle for

Un factorial es el producto de todos los enteros desde 1 hasta *N*. Por ejemplo, 5 factorial es

$$1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$$

En los textos de matemáticas, el factorial se indica usualmente con un signo de exclamación:

5! es cinco factorial.

MATLAB contiene una función interna para cálculo de factoriales, llamada **factorial**. Sin embargo, suponga que a usted le gustaría programar su propia función factorial llamada **fact**.

1. Establezca el problema.
   Crear una función llamada **fact** para calcular el factorial de cualquier número. Suponga entrada escalar.
2. Describa las entradas y salidas.

   Entrada Un valor escalar *N*

   Salida El valor de *N*!
3. Desarrolle un ejemplo a mano.

$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$$

4. Desarrolle una solución MATLAB.
   Primero desarrolle un diagrama de flujo (figura 8.17) para auxiliarse a planear su código.

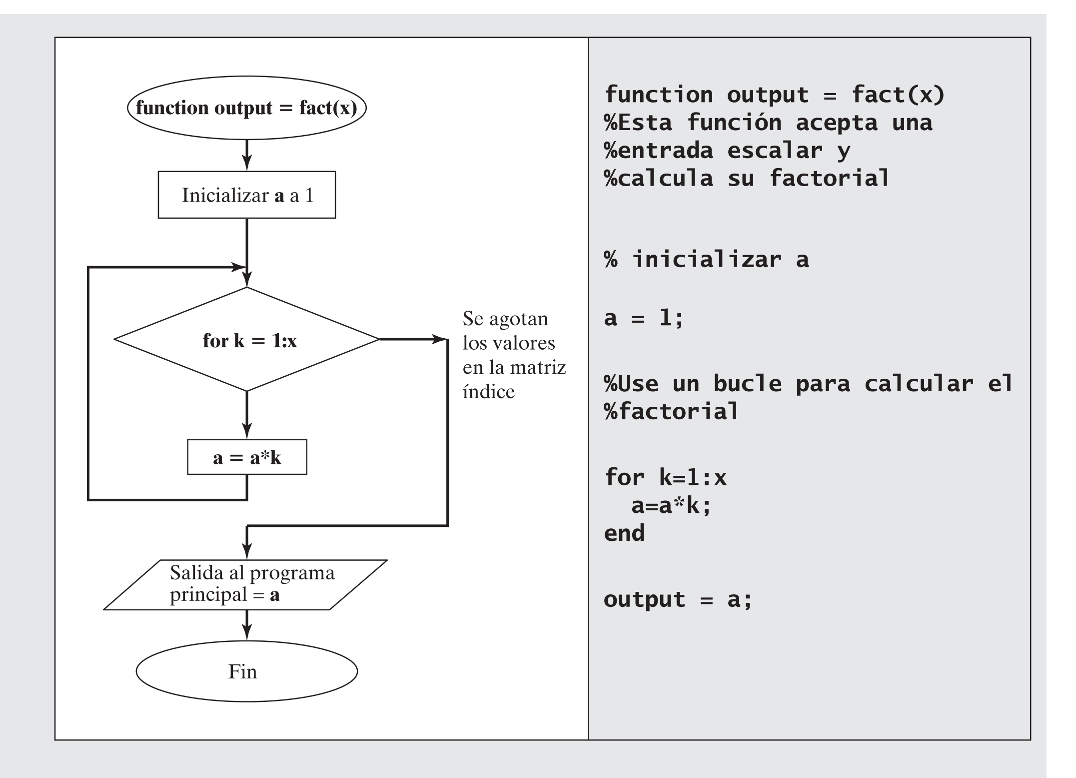


Figura 8.17
Diagrama de flujo para encontrar un factorial con el uso del bucle **for**.

5. Ponga a prueba la solución.
   Pruebe la función en la ventana de comandos:

```
fact(5)
ans =
   120
```

   Esta función sólo funciona si la entrada es un escalar. Si se ingresa un arreglo, el bucle **for** no se ejecuta y la función regresa un valor de 1:

```
x=1:10;
>> fact(x)
ans =
     1
```

   Puede agregar un enunciado **if** para confirmar que la entrada es un entero positivo y no un arreglo, como se muestra en el diagrama de flujo de la figura 8.18 y el código acompañante.

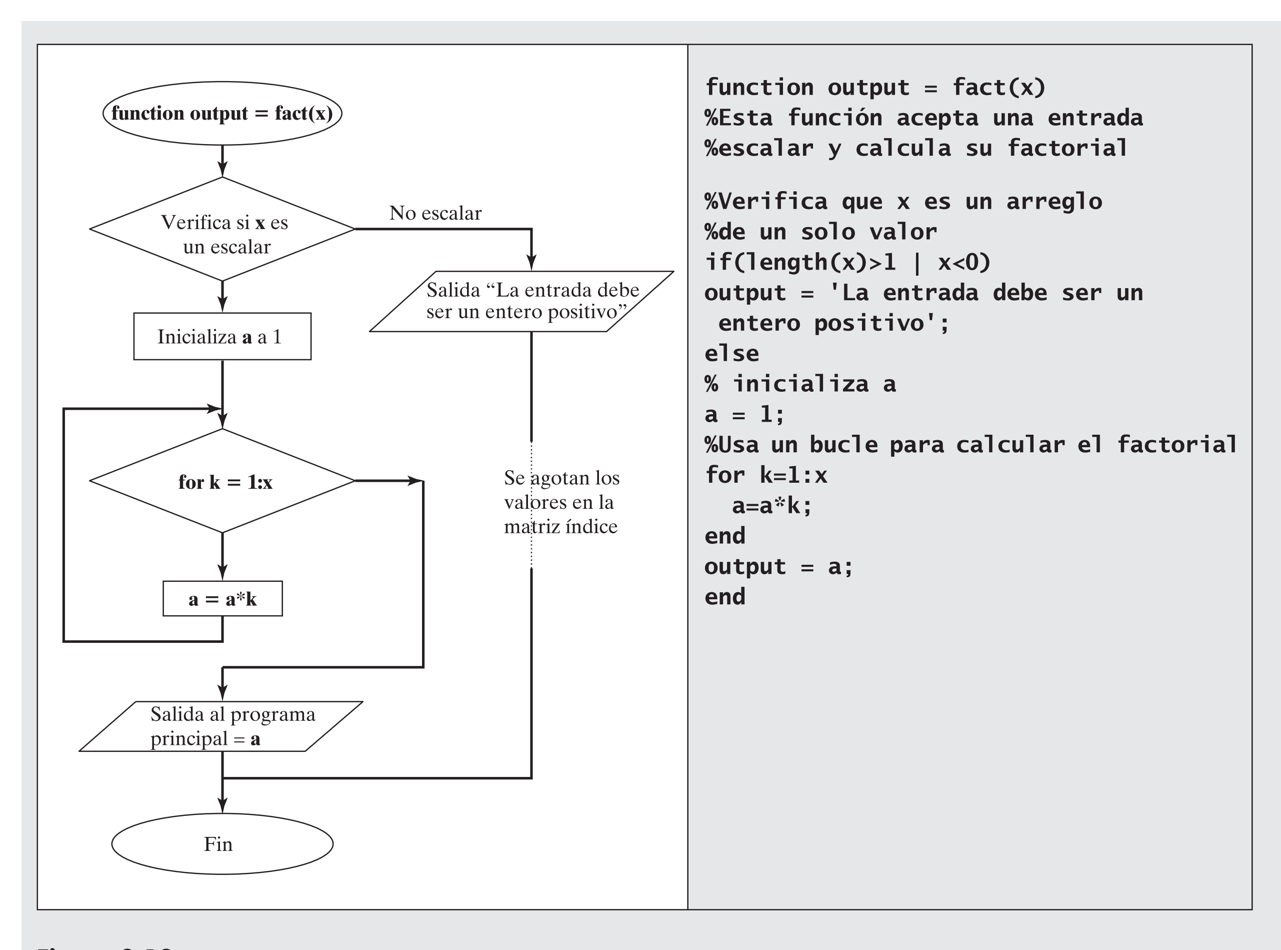


Figura 8.18
Diagrama de flujo para encontrar un factorial, incluida verificación de error.

Compruebe la nueva función en la ventana de comandos:

```
fact(-4)
ans =
La entrada debe ser un entero positivo

fact(x)
ans =
La entrada debe ser un entero positivo
```

Ejercicio de práctica 8.4

Use un bucle **for** para resolver los siguientes problemas:

1. Cree una tabla que convierta pulgadas a pies.
2. Considere la siguiente matriz de valores:

   x = [45,23,17,34,85,33]

   ¿Cuántos valores son mayores que 30? (Use un contador.)
3. Repita el problema 3, esta vez con el comando **find**.
4. Use un bucle **for** para sumar los elementos de la matriz en el problema 2. Compruebe sus resultados con la función **sum**. (Use la característica **help** si no sabe o recuerda cómo usar **sum**.)

8.5.2 Bucles while

Idea clave: use bucles while cuando no sepa cuántas veces deberá repetir una secuencia de comandos.

Los bucles **while** son similares a los bucles **for**. La gran diferencia es la forma en que MATLAB decide cuántas veces repetir el bucle. Los bucles **while** continúan hasta que se satisface algún criterio. El formato para un bucle **while** es

```
while  criterio
       comandos a ejecutar
end
```

He aquí un ejemplo:

```
k=0;
while k<3
  k=k+1
end
```

En este caso se inicializa un contador, **k**, antes del bucle. Entonces el bucle se repite en tanto **k** sea menor que 3. En cada ocasión, **k** aumenta por 1 a lo largo del bucle, de modo que el bucle se repite tres veces, lo que da

```
k =
   1
k =
   2
k =
   3
```

Se podría usar **k** como un número índice para definir una matriz o sólo como un contador. La mayoría de los bucles **for** también se pueden codificar como bucles **while**. Recuerde el bucle **for** de la sección 8.5.1 que se usó para calcular las tres primeras potencias de 5; el siguiente bucle **while** logra la misma tarea:

```
k=0;
while k<3
  k=k+1;
  a(k) = 5^k
end
```

El código regresa

```
a =
   5
a =
   5   25
a =
   5   25   125
```

Idea clave: cualquier problema que se pueda resolver con un bucle while también se podría resolver con un bucle for.

Cada vez que se pasa por el bucle, se agrega otro elemento a la matriz **a**.
Como otro ejemplo, inicialice primero **a**:

```
a = 0;
```

Entonces encuentre el primer múltiplo de 3 que es mayor que 10:

```
while(a<10)
        a = a + 3
end;
```

La primera vez que pase por el bucle, **a** es igual a 0, de modo que la comparación es verdadera. Se ejecuta el siguiente enunciado (**a = a + 3**) y el bucle se repite. Esta vez **a** es igual a 3 y la condición todavía es verdadera, de modo que la ejecución continúa. Sucesivamente, se tiene

```
a =
   3
a =
   6
a =
   9
a =
   12
```

La última vez que pasa por el bucle, **a** comienza como 9 y luego se convierte en 12 cuando se agrega 3 a 9. La comparación se realiza una última vez, pero, dado que **a** ahora es igual a 12 (que es mayor que 10), el programa salta al final del bucle **while** y ya no se repite.

Los bucles **while** también se pueden usar para contar cuántas veces una condición es verdadera al incorporar un enunciado **if**. Recuerde los puntajes de examen contados anteriormente en un bucle **for**. También se les puede contar con un bucle **while**:

```
scores = [76,45,98,97];
count = 0;
k=0;
while k<4
 k=k+1;
  if scores(k)>90
      count = count + 1;
  end
end
disp(count)
```

La variable **count** se usa para contar cuántos valores son mayores que 90. La variable **k** se usa para contar cuántas veces se ejecuta el bucle.

El diagrama de flujo básico para un bucle **while** es el mismo que el del bucle **for** (figura 8.19).

Sugerencia

La variable que se usa para controlar el bucle **while** se debe actualizar cada vez que pase por el bucle. Si no, generará un bucle interminable. Cuando un cálculo toma demasiado tiempo para completarse, puede confirmar que la computadora realmente trabaja en él al observar el indicador "busy", en la esquina inferior izquierda. Si quiere salir manualmente del cálculo, escriba **ctrl c**.

Sugerencia

Muchos textos y manuales de computadora indican la tecla control con el símbolo ^. Esto es confuso, en el mejor de los casos. El comando **^c** usualmente significa oprimir la tecla ctrl y la tecla c al mismo tiempo.

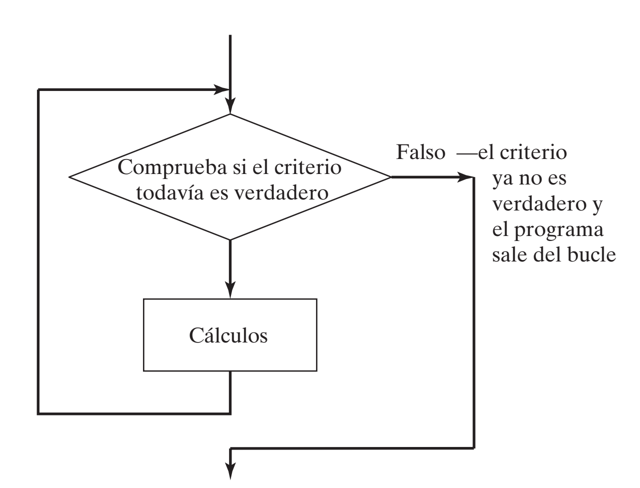


Figura 8.19
Diagrama de flujo para un bucle **while**.

EJEMPLO 8.7

Creación de una tabla para convertir grados a radianes con un bucle while

Tal como se usó un bucle **for** para crear una tabla para convertir grados a radianes en el ejemplo 8.5, se puede usar un bucle **while** para el mismo propósito.

1. Establezca el problema.
   Crear una tabla que convierta grados a radianes, desde 0 hasta 360 grados, en incrementos de 10 grados.
2. Describa las entradas y salidas.

   Entrada Un arreglo de valores de ángulo en grados

   Salida Una tabla de valores de ángulo tanto en grados como en radianes

3. Desarrolle un ejemplo a mano.
   Para 10 grados

$$\text{radianes} = (10)\frac{\pi}{180} = 0.1745$$

4. Desarrolle una solución MATLAB.
   Primero desarrolle un diagrama de flujo (figura 8.20) para ayudarse a planear su código.

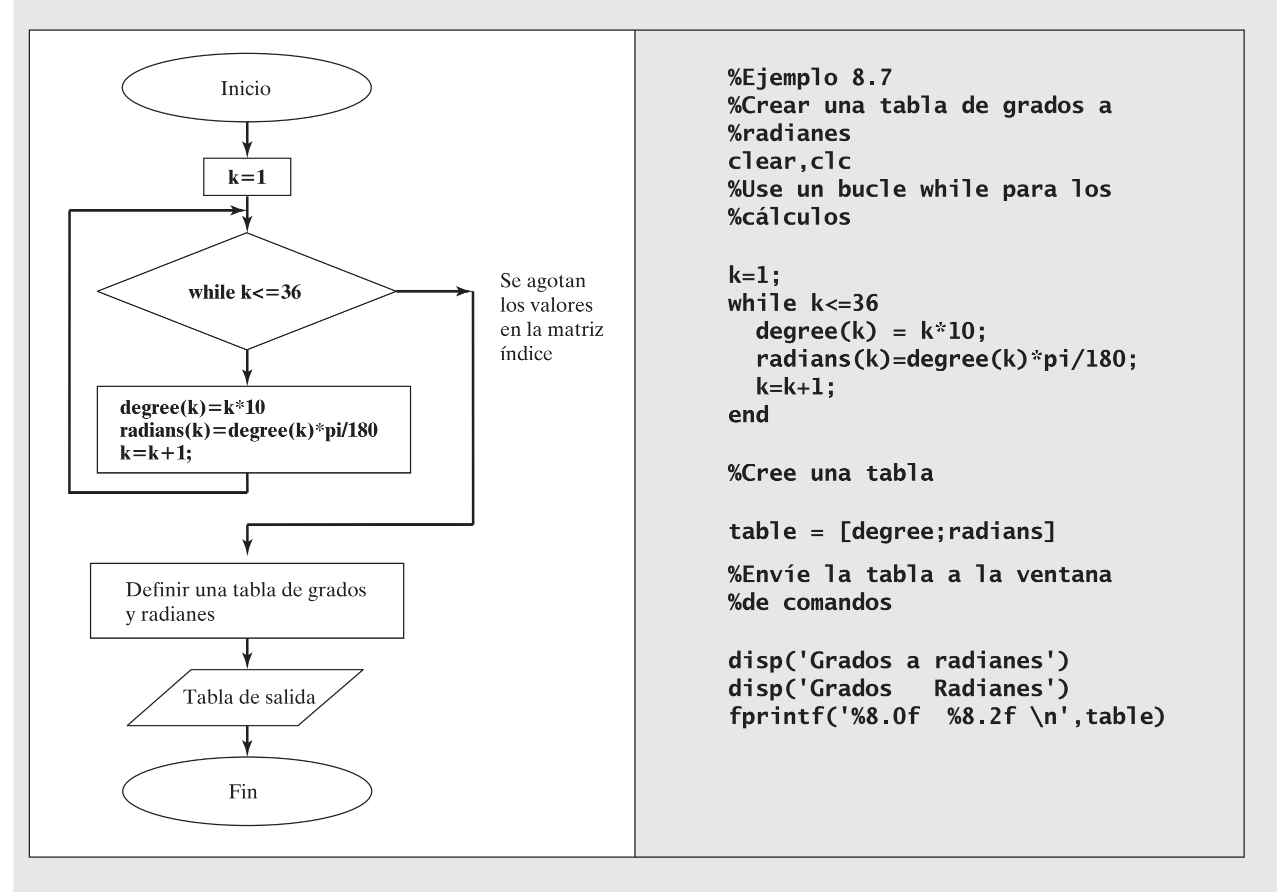


Figura 8.20
Diagrama de flujo para convertir grados a radianes con un bucle **while**.

La ventana de comandos despliega los siguientes resultados:

```
Grados a radianes
Grados   Radianes
   10       0.17
   20       0.35
   30       0.52   etc.
```

5. Ponga a prueba la solución.
   El valor para 10 grados calculado por MATLAB es el mismo que el del cálculo a mano.

EJEMPLO 8.8

Cálculo de factoriales con un bucle while

Cree una nueva función llamada **fact2** que use un bucle **while** para encontrar *N*! Incluya un enunciado **if** para verificar números negativos y confirmar que la entrada es un escalar.

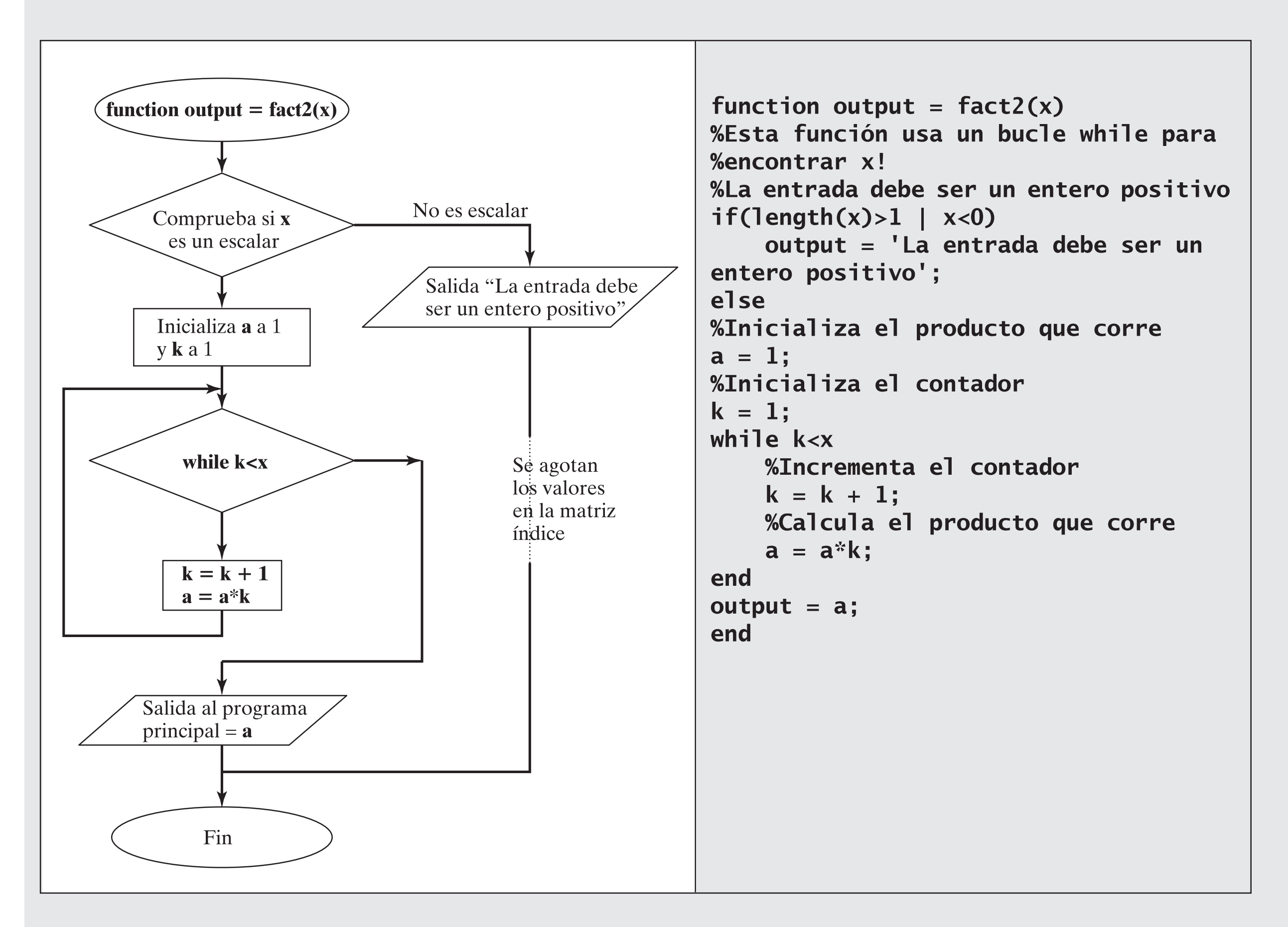
1. Establezca el problema.
   Crear una función llamada **fact2** para calcular el factorial de cualquier número.
2. Describa las entradas y salidas.

   Entrada Un valor escalar *N*

   Salida El valor de *N*!
3. Desarrolle un ejemplo a mano.

$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$$

4. Desarrolle una solución MATLAB.
   Desarrolle primero un diagrama de flujo (figura 8.21) para auxiliarse a planear su código.




```
function output = fact2(x)
%Esta función usa un bucle while para
%encontrar x!
%La entrada debe ser un entero positivo
if(length(x)>1 | x<0)
    output = 'La entrada debe ser un
entero positivo';
else
%Inicializa el producto que corre
a = 1;
%Inicializa el contador
k = 1;
while k<x
    %Incrementa el contador
    k = k + 1;
    %Calcula el producto que corre
    a = a*k;
end
output = a;
end
```

Figura 8.21
Diagrama de flujo para encontrar un factorial con un bucle **while**.

5. Ponga a prueba la solución.
   Pruebe la función en la ventana de comandos:

```
fact2(5)
ans =
   120

fact2(-10)
ans =
   La entrada debe ser un entero positivo

fact2([1:10])
ans =
   La entrada debe ser un entero positivo
```

Ejercicio de práctica 8.5

Use un bucle **while** para resolver los siguientes problemas:

1. Cree una tabla de conversión de pulgadas a pies.
2. Considere la siguiente matriz de valores:

   $x = [45,23,17,34,85,33]$

   ¿Cuántos valores son mayores que 30? (Use un contador.)
3. Repita el problema 3, esta vez con el comando **find**.
4. Use un bucle **while** para sumar los elementos de la matriz en el problema 2.

Compruebe sus resultados con la función **sum**. (Use la característica **help** si no sabe o recuerda cómo usar **sum**.)

8.5.3 Break y continue

El comando **break** se puede usar para terminar un bucle prematuramente (mientras que la comparación en la primera línea todavía es verdadera). Un enunciado **break** provocará la terminación de la parte más pequeña que encierre un bucle **while** o **for**. He aquí un ejemplo:

```
n=0;
while(n<10)
      n=n+1;
      a=input('Ingrese un valor mayor que 0:  ');
      if(a<=0)
         disp('Debe ingresar un número positivo ')
         disp(' Este programa terminará')
         break
      end
      disp('El log natural de este número es')
      disp(log(a))
   end
```

inicializar: definir un valor de partida para una variable que cambiará más tarde.

En este programa, el valor de **n** se inicializa afuera del bucle. Cada vez que se pasa por el bucle, el comando de entrada se usa para pedir un número positivo. El número se com-

prueba y, si es cero o negativo, se envía un mensaje de error a la ventana de comandos y el programa salta del bucle. Si el valor de **a** es positivo, el programa continúa y ocurre otro paso a través del bucle hasta que **n** finalmente es mayor que 10.

El comando **continue** es similar a **break**; sin embargo, en lugar de terminar el bucle, el programa sólo salta al paso siguiente:

```
n=0;
while(n<10)
      n=n+1;
      a=input('Ingrese un valor mayor que 0:  ');
      if(a<=0)
          disp('Debe ingresar un número positivo ')
          disp(' Intente de nuevo')
          continue
      end
      disp('El log natural de este número es')
      disp(log(a))
    end
```

En este ejemplo, si ingresa un número negativo, el programa le permite intentar de nuevo, hasta que el valor de **n** finalmente sea mayor que 10.

8.5.4 Cómo mejorar la eficiencia de los bucles

Idea clave: los bucles, por lo general, son menos eficientes que los cálculos vectorizados.

En general, usar un bucle **for** (o un bucle **while**) es menos eficiente en MATLAB que usar operaciones de arreglos. Se puede probar esta afirmación al cronometrar la multiplicación de los elementos en un arreglo largo. Primero se crea una matriz **A** que contiene 40,000 unos. El comando **ones** crea una matriz $n \times n$ de unos:

```
ones(200);
```

El resultado es una matriz 200×200 de unos. Ahora se pueden comparar los resultados de multiplicar cada elemento por π, usando primero una multiplicación de arreglo y luego un bucle **for**. Puede cronometrar los resultados con el uso de la función **clock** y la función **etime**, que mide el tiempo transcurrido. Si tiene una computadora rápida, es posible que requiera usar un arreglo más largo. La estructura del código de temporización es

```
t0 = clock;
código a temporizar
etime (clock, t0)
```

La función clock pregunta al reloj de la computadora el tiempo actual. La función **etime** compara el tiempo actual con el tiempo inicial y resta los dos valores para dar el tiempo transcurrido.

Para el problema,

```
clear, clc
A=ones(200);   %Crea una matriz 200 x 200 de unos
t0=clock;
     B=A*pi;
time = etime(clock, t0);
```

da un resultado de

```
time =
      0
```

El cálculo de arreglo toma 0 segundos, lo que simplemente significa que ocurrió muy rápidamente. Cada vez que corra estas líneas de código debe obtener una respuesta diferente. Las funciones **clock** y **etime** que se usan aquí miden cuánto tiempo trabajó el CPU entre recibir las peticiones de temporización original y final. Sin embargo, el CPU hace otras cosas además del problema: en un mínimo, realiza tareas del sistema y puede correr otros programas en segundo plano.

Para medir el tiempo requerido para realizar el mismo cálculo con un bucle se necesita limpiar la memoria y volver a crear el arreglo de unos:

```
clear
A=ones(200);
```

Esto asegura que se comparan los cálculos desde el mismo punto de partida. Ahora se codifica

```
t0=clock;
   for k=1:length(A(:))
      B(k)=A(k)*pi;
   end
time = etime(clock, t0)
```

lo que produce el resultado

```
time =
       69.6200
```

¡Toma casi 70 segundos realizar el mismo cálculo! (Esto fue en la computadora del autor, su resultado dependerá de la máquina que use.) El número de iteraciones a través del bucle **for** se determinó mediante hallar cuántos elementos hay en **A**. Esto se logró con el comando **length**. Recuerde que **length** regresa la dimensión de arreglo más grande, que es 200 para el arreglo y no es el que se quería. Para encontrar el número total de elementos se usó el operador dos puntos (:) para representar **A** como una sola lista de 40,000 elementos de largo, y luego se usó **length**, que regresó 40,000. Cada vez que pasa por el bucle **for**, se agrega un nuevo elemento a la matriz **B**. Éste es el paso que toma todo el tiempo. Se puede reducir el tiempo requerido para este cálculo al crear primero la matriz **B** y luego sustituir los valores uno a la vez. El código es

```
clear
A=ones(200);
t0=clock;
%Crea una matriz B de unos
   B = A;
   for k=1:length(A(:))
     B(k)=A(k)*pi;
   end
time = etime(clock, t0)
```

que da como resultado

```
time =
       0.0200
```

Obviamente, ésta es una enorme mejoría. Podría ver una diferencia incluso más grande entre el primer ejemplo, una simple multiplicación de los elementos de un arreglo, y el último ejemplo, si crea una matriz más grande. En contraste, el ejemplo intermedio, en el que no se inicializó **B**, tomaría una cantidad prohibitiva de tiempo en ejecutar.

MATLAB también incluye un conjunto de comandos llamados **tic** y **toc** que se pueden usar en forma similar a las funciones **clock** y **etime** para cronometrar un trozo de código. Por lo tanto, el código

```
clear
A=ones(200);
tic
     B = A;
     for k=1:length(A(:))
       B(k)=A(k)*pi;
     end
toc
```

regresa

```
El tiempo transcurrido es 0.140000 segundos.
```

Se esperaba la diferencia en el tiempo de ejecución, pues la computadora está ocupada haciendo diferentes tareas en segundo plano cada vez que el programa se ejecuta. Como con **clock/etime**, los comandos **tic/toc** miden el tiempo transcurrido, no el tiempo dedicado sólo a la ejecución de este programa.

Sugerencia

Asegúrese de suprimir cálculos intermedios cuando use un bucle. Imprimir dichos valores a la pantalla aumentará enormemente la cantidad de tiempo de ejecución. Si tiene valor, repita el ejemplo anterior, pero borre los puntos y coma dentro del bucle sólo para corroborar esta afirmación. No olvide que puede detener la ejecución del programa con **ctrl c**.

RESUMEN

Secciones de código de computadora se pueden categorizar como secuencias, estructuras de selección y estructuras de repetición. Las secuencias son listas de instrucciones que se ejecutan en orden. Las estructuras de selección permiten al programador definir criterios (enunciados condicionales) que el programa usa para elegir rutas de ejecución. Las estructuras de repetición definen bucles en los que una secuencia de instrucciones se repite hasta que se satisface algún criterio (también definidos mediante enunciados condicionales).

MATLAB usa los operadores relacionales matemáticos estándar, como mayor que (>) y menor que (<). La forma del operador no igual a (~=) usualmente no se ve en los textos de

matemáticas. Además, MATLAB incluye operadores lógicos como *and* (&) y *or* (|). Estos operadores se usan en enunciados condicionales, lo que permite a MATLAB tomar decisiones acerca de cuáles porciones del código ejecutar.

El comando **find** es único para MATLAB y debe ser la función condicional primaria usada en su programación. Este comando permite al usuario especificar una condición mediante el uso de operadores lógicos y relacionales. Entonces el comando se usa para identificar elementos de una matriz que satisface la condición.

Aunque se pueden usar los comandos **if**, **else** y **elseif** tanto para variables escalares como matriciales, son útiles principalmente para escalares. Estos comandos permiten al programador identificar rutas de cómputo alternativas sobre la base de los resultados de enunciados condicionales.

Los bucles **for** se usan principalmente cuando el programador sabe cuántas veces se debe ejecutar una secuencia de comandos. Los bucles **while** se usan cuando los comandos se deben ejecutar hasta que se satisface una condición. La mayoría de los problemas se pueden estructurar de modo que sean adecuados los bucles **for** o **while**.

Los enunciados **break** y **continue** se usan para salir prematuramente de un bucle. Por lo general, se usan en conjunto con enunciados **if**. El comando **break** provoca un salto de un bucle y la ejecución del resto del programa. El comando **continue** salta la ejecución del paso actual en el bucle, pero permite al bucle continuar hasta completar la satisfacción de criterio.

La vectorización del código MATLAB le permite ejecutar mucho más eficientemente y, por tanto, más rápidamente. Los bucles en particular se deben evitar en MATLAB. Cuando los bucles son inevitables, se pueden mejorar mediante la definición de variables "dummy" con valores de marcador de posición, como unos o ceros. Luego, dichos marcadores de posición se pueden sustituir en el bucle. Hacer esto resultará en mejoras significativas en el tiempo de ejecución, un hecho que se puede confirmar con experimentos de temporización.

Las funciones **clock** y **etime** se usan para consultar el reloj de la computadora y luego determinar el tiempo que se requiere para ejecutar trozos de código. El tiempo calculado es el tiempo "transcurrido". Durante este tiempo, la computadora no sólo corre un código MATLAB, sino también ejecuta tareas de fondo y funciones de mantenimiento. Las funciones **tic** y **toc** realizan una tarea similar. Para comparar el tiempo de ejecución de diferentes opciones de código se pueden usar las funciones o **tic/toc** o **clock/etime**.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona y describe brevemente todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres especiales	
<	menor que
<=	menor que o igual a
>	mayor que
>=	mayor que o igual a
==	igual a
~=	no igual a
&	and
|	or
~	not

Comandos y funciones	
`all`	verifica si un criterio se satisface por todos los elementos de un arreglo
`any`	verifica si un criterio se satisface por alguno de los elementos en el arreglo
`break`	provoca que la ejecución de un bucle se termine
`case`	ordena las respuestas
`clock`	determina el tiempo actual en el reloj del CPU
`continue`	termina el paso actual a través de un bucle, pero procede al siguiente paso
`else`	define la ruta si el resultado de un enunciado **if** es falso
`elseif`	define la ruta si el resultado de un enunciado **if** es falso y especifica una nueva prueba lógica
`end`	identifica el final de una estructura de control
`etime`	encuentra el tiempo transcurrido
`find`	determina cuáles elementos en una matriz satisfacen el criterio de entrada
`for`	genera una estructura bucle
`if`	verifica una condición que resulta en verdadero o en falso
`menu`	crea un menú para usar como vehículo de entrada
`ones`	crea una matriz de unos
`otherwise`	parte de la estructura de selección de caso
`switch`	parte de la estructura de selección de caso
`tic`	inicia una secuencia de cronometrado
`toc`	detiene una secuencia de cronometrado
`while`	genera una estructura bucle

TÉRMINOS CLAVE

bucle (loop)
condición lógica
estructura de control
índice
operador lógico
operador relacional
repetición
secuencia
selección
subíndice
variable local

PROBLEMAS

Operadores lógicos: find

8.1 Un sensor que monitorea la temperatura de un tubo de calentamiento de un patio trasero registra los datos que se muestran en la tabla 8.5.

(a) La temperatura nunca debe superar los 105 °F. Use la función **find** para encontrar los números índice de las temperaturas que superan la temperatura máxima permisible.
(b) Use la función **length** con los resultados de la parte (a) para determinar cuántas veces se superó la temperatura máxima permisible.
(c) Determine en qué momentos la temperatura superó la temperatura máxima permisible con los números índice que se encontraron en la parte (a).
(d) La temperatura nunca debe ser menor que 102 °F. Use la función **find** junto con la función **length** par determinar cuántas veces la temperatura fue menor que la temperatura mínima permisible.

(e) Determine en qué momento la temperatura fue menor que la temperatura mínima permisible.
(f) Determine en qué momento la temperatura estuvo dentro de los límites permisibles (es decir: entre 102 °F y 105 °F, inclusive).
(g) Use la función **max** para determinar la temperatura máxima alcanzada y el momento en que ocurrió.

8.2 La altura de un cohete (en metros) se puede representar mediante la siguiente ecuación:

$$\text{height} = 2.13t^2 - 0.0013t^4 + 0.000034t^{4.751}$$

Cree un vector de valores de tiempo (t) desde 0 hasta 100 a intervalos de 2 segundos.

(a) Use la función **find** para determinar cuándo el cohete golpea el suelo hasta dentro de 2 segundos. (*Sugerencia:* el valor de **altura** será positivo para todos los valores hasta que el cohete golpee el suelo.)
(b) Use la función **max** para determinar la altura máxima del cohete y el tiempo correspondiente.
(c) Cree una gráfica con t en el eje horizontal y altura en el eje vertical para tiempos hasta que el cohete golpee el suelo. Asegúrese de agregar un título y etiquetas de eje.*

8.3 Los motores de cohete sólidos se usan como propulsor auxiliar (booster) para el transbordador espacial, en vehículos de lanzamiento de satélites y en sistemas de armas (véase la figura P8.3). El propelente es una combinación sólida de combustible y oxidante, de consistencia gomosa. Para el transbordador espacial, el componente combustible es aluminio y el oxidante es percloruro de amonio, que se mantienen juntos con un "pegamento" de resina epóxica. La mezcla propelente se vierte en un alojamiento del motor y se permite que la resina cure bajo condiciones controladas. Puesto que los motores son extremadamente grandes, se moldean en segmentos, cada uno de los cuales requiere muchos "lotes" de propelente para llenarse. (¡Cada motor contiene más de 1.1 millones de libras de propelente!) Este proceso de curado-moldeado es sensible a temperatura, humedad y presión. Si las condiciones no son las correctas, el combustible podría incendiarse o las propiedades granulosas del propelente (es decir, su forma; el término *grano* se tomó prestado de artillería) podrían degradarse. Los motores de cohete sólidos son extremadamente costosos y obviamente deben funcionar bien en todo momento, o los resultados podrían ser desastrosos. Los fracasos extremadamente públicos pueden destruir una compañía y provocar pérdidas de vidas humanas, de datos y equipos científicos irremplazables. Los procesos actuales se monitorean y controlan con mucha precisión. Sin embargo, para propósitos de enseñanza, considere estos criterios generales:

La temperatura debe permanecer entre 115 °F y 125 °F.
La humedad debe permanecer entre 40 y 60 por ciento.
La presión debe permanecer entre 100 y 200 torr.

Imagine que los datos de la tabla 8.6 se recopilaron durante un proceso de curado-moldeado.

(a) Use el comando **find** para determinar cuáles lotes satisfacen y cuáles no el criterio de temperatura.

Tabla 8.5 Datos de temperatura de tubo de calentamiento

Hora del día	Temperatura °F
0:00 A.M.	100
1:00 A.M.	101
2:00 A.M.	102
3:00 A.M.	103
4:00 A.M.	103
5:00 A.M.	104
6:00 A.M.	104
7:00 A.M.	105
8:00 A.M.	106
9:00 A.M.	106
10:00 A.M.	106
11:00 A.M.	105
12:00 P.M.	104
1:00 P.M.	103
2:00 P.M.	101
3:00 P.M.	100
4:00 P.M.	99
5:00 P.M.	100
6:00 P.M.	102
7:00 P.M.	104
8:00 P.M.	106
9:00 P.M.	107
10:00 P.M.	105
11:00 P.M.	104
12:00 A.M.	104

*Tomado de Etter, Kuncicky y Moore, *Introducción a MATLAB 7* (Upper Saddle River, NJ: Pearson/Prentice Hall, 2005).



Figura P8.3
Booster de cohete sólido para un misil Titán. (Cortesía de la NASA.)

Tabla 8.6 Datos curado-moldeado

Número de lote	Temperatura °F	Humedad %	Presión psi
1	116	45	110
2	114	42	115
3	118	41	120
4	124	38	95
5	126	61	118

(b) Use el comando **find** para determinar cuáles lotes satisfacen y cuáles no el criterio de humedad.
(c) Use el comando **find** para determinar cuáles lotes satisfacen y cuáles no el criterio de presión.
(d) Use el comando **find** para determinar cuáles lotes fallan por cualquier razón y cuáles aprueban.
(e) Use sus resultados de las preguntas anteriores, junto con el comando **length**, para determinar qué porcentaje de motores aprobó o falló sobre la base de cada criterio y para determinar la tasa de aprobación total.

8.4 Dos gimnastas compiten entre sí. Sus calificaciones se muestran en la tabla 8.7.

(a) Escriba un programa que use **find** para determinar cuántos eventos ganó cada gimnasta.
(b) Use la función **mean** para determinar la puntuación promedio de cada gimnasta.

8.5 Cree una función llamada **f** que satisfaga los siguientes criterios:

$$\text{Para valores de } x > 2, f(x) = x^2$$

$$\text{Para valores de } x \leq 2, f(x) = 2x$$

Tabla 8.7 Puntajes de gimnastas

Evento	Gimnasta 1	Gimnasta 2
Pommel horse	9.821	9.700
Vault	9.923	9.925
Floor	9.624	9.83
Rings	9.432	9.987
High bar	9.534	9.354
Parallel bars	9.203	9.879

Grafique sus resultados para valores de x desde -3 hasta 5. Elija su espaciamiento para crear una curva suave. Debe notar un rompimiento en la curva en $x = 2$.

8.6 Cree una función llamada g que satisfaga los siguientes criterios:

$$\begin{aligned} \text{Para } x < -\pi, &\quad f(x) = -1 \\ \text{Para } x \geq -\pi \text{ y } x \leq \pi, &\quad f(x) = \cos(x) \\ \text{Para } x > \pi, &\quad f(x) = -1 \end{aligned}$$

Grafique sus resultados para valores de x desde $-2\ \pi$ hasta $+2\ \pi$. Elija su espaciamiento para crear una curva suave.

8.7 Un archivo llamado **temp.dat** contiene información recopilada de un conjunto de termocoples. Los datos en el archivo se muestran en la tabla 8.8. La primera columna incluye mediciones de tiempo (una para cada hora del día) y las restantes

Tabla 8.8 Datos de temperatura

Hora	Temp1	Temp2	Temp3
1	68.70	58.11	87.81
2	65.00	58.52	85.69
3	70.38	52.62	71.78
4	70.86	58.83	77.34
5	66.56	60.59	68.12
6	73.57	61.57	57.98
7	73.57	67.22	89.86
8	69.89	58.25	74.81
9	70.98	63.12	83.27
10	70.52	64.00	82.34
11	69.44	64.70	80.21
12	72.18	55.04	69.96
13	68.24	61.06	70.53
14	76.55	61.19	76.26
15	69.59	54.96	68.14
16	70.34	56.29	69.44
17	73.20	65.41	94.72
18	70.18	59.34	80.56
19	69.71	61.95	67.83
20	67.50	60.44	79.59
21	70.88	56.82	68.72
22	65.99	57.20	66.51
23	72.14	62.22	77.39
24	74.87	55.25	89.53

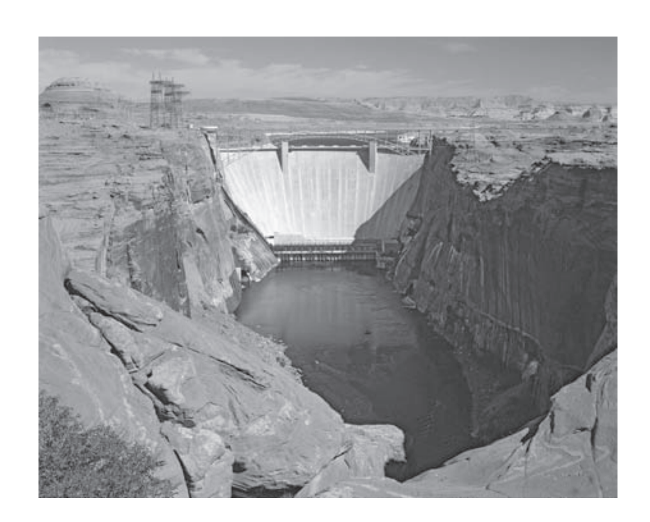


Figura P8.8
Presa del cañón Glen en el lago Powell. (Cortesía de Getty Images, Inc.)

columnas corresponden a mediciones de temperatura en diferentes puntos en un proceso.

(a) Escriba un programa que imprima los números índice (filas y columnas) de valores de datos de temperatura mayores que 85.0. (*Sugerencia:* necesitará usar el comando **find**.)
(b) Encuentre los números índice (filas y columnas) de valores de datos de temperatura menores que 65.0.
(c) Encuentre la temperatura máxima en el archivo y los correspondientes valores de hora y número de termocople.

8.8 La cuenca fluvial del río Colorado cubre partes de siete estados occidentales. Una serie de presas se construyeron en el río Colorado y sus tributarios para almacenar agua que corre y generar energía hidroeléctrica de bajo costo. (Véase la figura P8.8.) La habilidad para regular el flujo de agua hizo posible el crecimiento de la agricultura y la población en estos estados áridos y desérticos. Incluso durante periodos de sequía intensa, para los estados de la cuenca está disponible una fuente confiable y estable de agua y electricidad. El lago Powell es uno de dichos depósitos. El archivo **lake_powell.dat** contiene datos acerca del nivel del agua en el depósito para los cinco años desde 2000 hasta 2004. Dichos datos se muestran en la tabla 8.9. Use la tabla para resolver los siguientes problemas:

(a) Determine la elevación promedio del nivel del agua para cada año y para el periodo de cinco años durante el que se recopilaron los datos.
(b) Determine cuántos meses se superó cada año el promedio global para el periodo de cinco años.
(c) Cree un reporte que mencione el mes y el año para cada uno de los meses que superaron el promedio global.
(d) Determine la elevación promedio del agua para cada mes para el periodo de cinco años.

Estructuras if

8.9 Cree un programa que conmine al usuario a ingresar un valor escalar de temperatura. Si la temperatura es mayor que 98.6 °F, envíe un mensaje a la ventana de comandos que diga al usuario que tiene fiebre.

8.10 Cree un programa que primero conmine al usuario a ingresar un valor de **x** y luego lo conmine a ingresar un valor para **y**. Si el valor de **x** es mayor que el valor de **y**, envíe

Tabla 8.9 Datos de nivel de agua para el lago Powell, medidos en pies sobre el nivel del mar

	2000	2001	2002	2003	2004
Enero	3680.12	3668.05	3654.25	3617.61	3594.38
Febrero	3678.48	3665.02	3651.01	3613	3589.11
Marzo	3677.23	3663.35	3648.63	3608.95	3584.49
Abril	3676.44	3662.56	3646.79	3605.92	3583.02
Mayo	3676.76	3665.27	3644.88	3606.11	3584.7
Junio	3682.19	3672.19	3642.98	3615.39	3587.01
Julio	3682.86	3671.37	3637.53	3613.64	3583.07
Agosto	3681.12	3667.81	3630.83	3607.32	3575.85
Septiembre	3678.7	3665.45	3627.1	3604.11	3571.07
Octubre	3676.96	3663.47	3625.59	3602.92	3570.7
Noviembre	3674.93	3661.25	3623.98	3601.24	3569.69
Deciembre	3671.59	3658.07	3621.65	3598.82	3565.73

un mensaje a la ventana de comandos que le diga al usuario que **x > y**. Si **x** es menor que o igual a **y**, envíe un mensaje a la ventana de comandos que le diga al usuario que **y >= x**.

8.11 Las funciones seno inverso (**asin**) y coseno inverso (**acos**) son válidas sólo para entradas entre −1 y +1, porque tanto el seno como el coseno tienen valores sólo entre −1 y +1 (figura P8.11). MATLAB interpreta el resultado de **asin** o **acos** para un valor fuera del rango como un número complejo. Por ejemplo, se puede tener

```
acos(-2)
ans =
  3.1416 - 1.3170i
```

que es un resultado matemático cuestionable. Cree una función llamada **my_asin** que acepte un solo valor de **x** y verifique si está entre −1 y +1 (**−1 <= x <= 1**). Si **x** está fuera del rango, envíe un mensaje de error a la pantalla. Si está adentro del rango permisible, regrese el valor de **asin**.

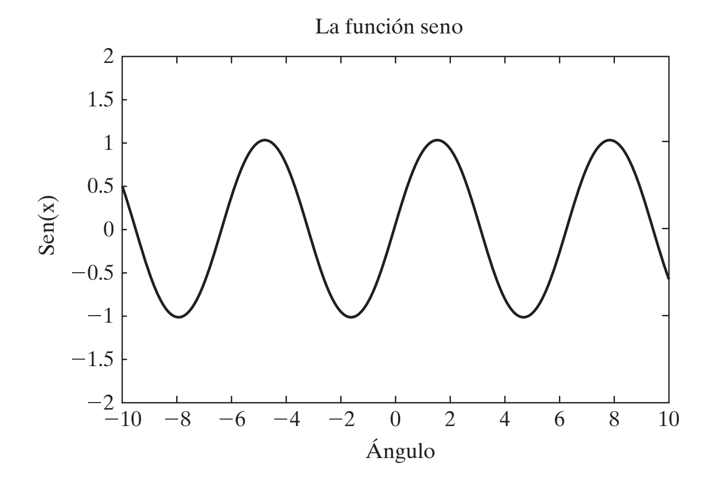


Figura P8.11
La función seno varía entre −1 y +1. Por tanto, el seno inverso (**asin**) no está definido para valores mayores que 1 y menores que −1.

8.12 Cree un programa que conmine al usuario a ingresar un valor escalar para la temperatura del aire exterior. Si la temperatura es igual a o sobre 80 °F, envíe un mensaje a la ventana de comandos que le diga al usuario que vista pantalones cortos. Si la temperatura está entre 60 y 80 °F, envíe un mensaje a la ventana de comandos que le diga al usuario que es un hermoso día. Si la temperatura es igual a o por abajo de 60 °F, envíe un mensaje a la ventana de comandos que le diga al usuario que vista una chamarra o abrigo.

8.13 Suponga que la siguiente matriz representa el número de sierras ordenadas por su compañía cada mes durante el último año.

```
saws = [1,4,5,3,7,5,3,10,12,8, 7, 4]
```

Todos los números deben ser cero o positivos.

(a) Use un enunciado **if** para verificar si alguno de los valores en la matriz es inválido. (Evalúe toda la matriz una vez en un solo enunciado **if**.) Envíe a la pantalla el mensaje "Todos válidos" o "Número inválido encontrado", dependiendo de los resultados de su análisis.

(b) Cambie la matriz **saws** para incluir al menos un número negativo y verifique su programa para asegurarse de que funciona para ambos casos.

8.14 La mayoría de las compañías grandes alientan a los empleados a ahorrar igualando sus contribuciones a un plan 401(k). El gobierno limita cuánto puede una persona ahorrar en dichos planes, porque blindan el ingreso de impuestos hasta que el dinero se retira durante su jubilación. La cantidad que la persona puede ahorrar se liga a su ingreso, ya que es la cantidad que puede aportar su empleador. El gobierno le permitirá ahorrar cantidades adicionales sin el beneficio impositivo. Estos planes cambian de año en año, de modo que este ejemplo sólo es un escenario "qué sucede si...".

Suponga que la Quality Widget Company tiene los planes de ahorro descritos en la tabla 8.10. Cree una función que encuentre la aportación anual total a su plan de ahorros, con base en su salario y el porcentaje que usted aporta.

Tabla 8.10 Plan de ahorro de Quality Widget Company

Ingreso	Máximo que puede ahorrar libre de impuestos	Máximo que igualará la compañía
Hasta $30,000	10%	10%
Entre $30,000 y $60,000	10%	10% de los primeros $30,000 y 5% de la cantidad arriba de $30,000
Entre $60,000 y $100,000	10% de los primeros $60,000 y 8% de la cantidad sobre $60,000	10% de los primeros $30,000 y 5% de la cantidad entre $30,000 y $60,000; nada por el resto arriba de $60,000
Arriba de $100,000	10% de los primeros $60,000 y 8% de la cantidad entre $60,000 y $100,000; nada sobre la cantidad arriba de $100,000	Nada: los empleados con remuneraciones altas están exentos de este plan y en vez de ello participan en opciones de acciones

Recuerde: la contribución total consiste en la aportación del empleado y la aportación de la compañía.

Switch/case

8.15 Con la finalidad de tener una figura geométrica cerrada compuesta de líneas rectas (figura P8.15), los ángulos en la figura deben sumar

$$(n - 2)(180 \text{ grados})$$

donde n es el número de lados.

(a) Pruebe este enunciado usted mismo mediante la creación de un vector llamado n desde 3 hasta 6 y calcule la suma de ángulo a partir de la fórmula. Compare lo que sabe de geometría con su respuesta.

(b) Escriba un programa que conmine al usuario a ingresar uno de los siguientes:

triángulo
cuadrado
pentágono
hexágono

Use la entrada para definir el valor de n mediante una estructura **switch/case**; luego use n para calcular la suma de los ángulos interiores de la figura.

(c) Reformule su programa desde la parte (b) de modo que use un menú.

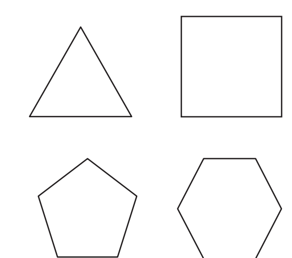


Figura P8.15
Polígonos regulares.

8.16 En la Universidad de Utah, cada especialización en ingeniería requiere un número diferente de créditos para graduación. Por ejemplo, en 2005, los requisitos para algunos de los departamentos fueron los siguientes:

Ingeniería civil	130
Ingeniería química	130
Ingeniería en computación	122
Ingeniería eléctrica	126.5
Ingeniería mecánica	129

Conmine al usuario a seleccionar un programa de ingeniería desde un menú. Use una estructura **switch/case** para enviar el número mínimo de créditos requeridos para graduación de vuelta a la ventana de comandos.

8.17 La forma más sencilla de dibujar una estrella en MATLAB es usar coordenadas polares. Simplemente necesita identificar puntos sobre la circunferencia de un círculo y dibujar líneas entre dichos puntos. Por ejemplo, para dibujar una estrella de cinco puntas, comience en la parte superior del círculo ($\theta = \pi/2$, $r = 1$) y vaya en sentido contrario al de las manecillas del reloj (figura P8.17).

Conmine al usuario a especificar una estrella de cinco puntas o de seis puntas mediante un menú. Luego cree la estrella en una ventana de figura MATLAB. Note que una estrella de seis puntas se hace con tres triángulos y requiere una estrategia diferente de la que usó para crear una estrella de cinco puntas.

Estructuras de repetición: bucles

8.18 Use un bucle **for** para sumar los elementos del siguiente vector:

$$x = [1,23,43,72,87,56,98,33]$$

Compruebe su respuesta con la función **sum**.

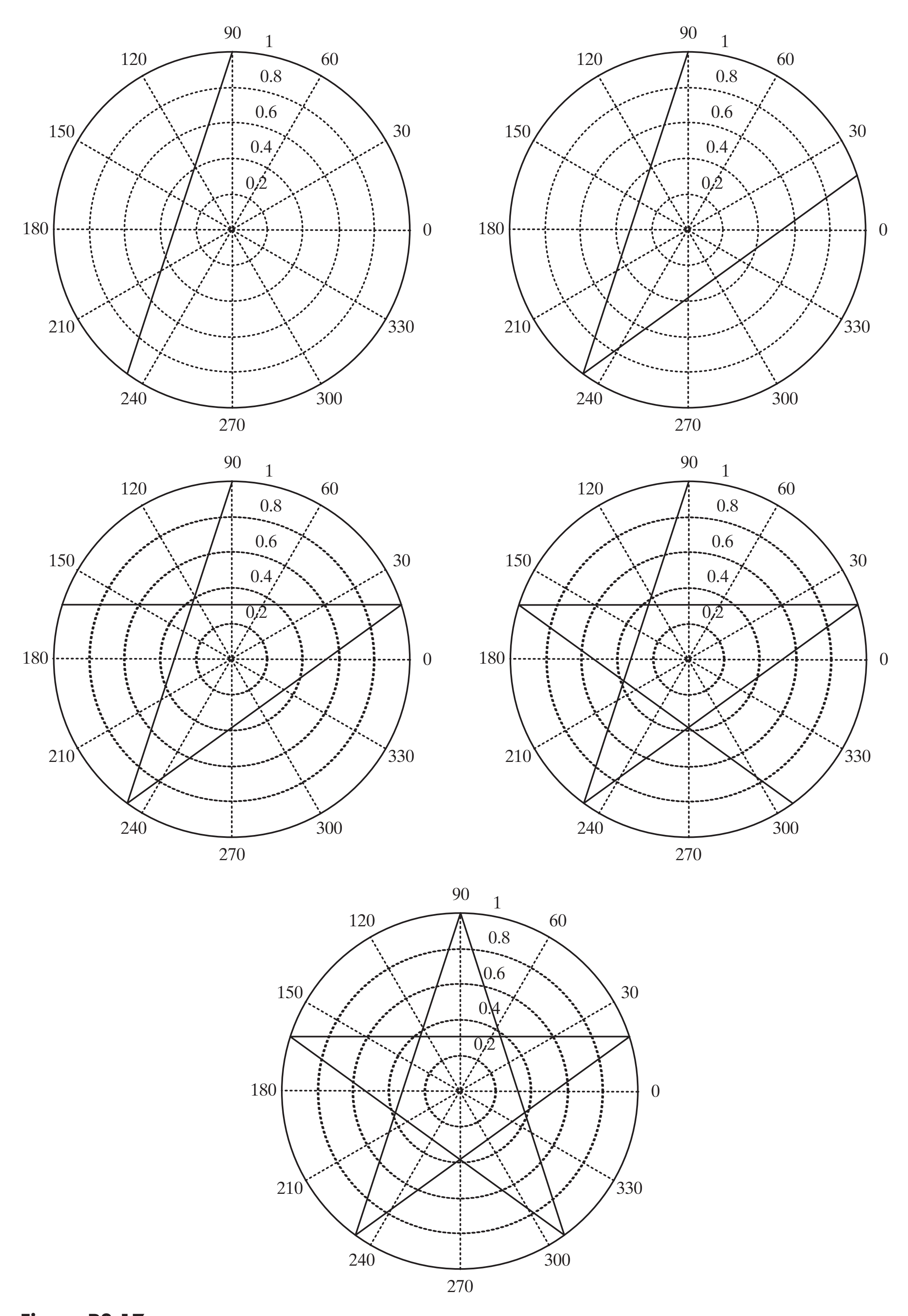


Figura P8.17

Pasos que se requieren para dibujar una estrella de cinco puntas en coordenadas polares.

8.19 Repita el problema anterior, pero esta vez use un bucle **while**.

8.20 Una secuencia de Fibonacci está compuesta de elementos creados al sumar los dos elementos previos. La secuencia de Fibonacci más simples comienza con 1, 1 y procede del modo siguiente:

1, 1, 2, 3, 5, 8, 13, etcétera

Sin embargo, una secuencia de Fibonacci se puede crear con cualesquiera dos números iniciales. Las secuencias de Fibonacci aparecen regularmente en la naturaleza. Por ejemplo, la concha del nautilus con cámaras (figura P8.20) crece en concordancia con una secuencia de Fibonacci.

Conmine al usuario a ingresar los dos primeros números en una secuencia de Fibonacci y el número total de elementos solicitados para la secuencia. Encuentre la secuencia al usar un bucle **for**. Ahora grafique sus resultados en una gráfica **polar**. Use el número de elemento para el ángulo y el valor del elemento en la secuencia para el radio.



Figura P8.20
Nautilus con cámaras. (Cortesía de Dorling Kindersley.)

8.21 Repita el problema anterior, esta vez con un bucle **while**.

8.22 Una propiedad interesante de una secuencia de Fibonacci es que la razón de los valores de miembros adyacentes de la secuencia se aproxima a un número llamado "razón dorada" o Φ (fi). Cree un programa que acepte los primeros dos números de una secuencia de Fibonacci como entrada de usuario y luego calcule valores adicionales en la secuencia hasta que la razón de valores adyacentes converja hasta dentro de 0.001. Puede hacer esto con un bucle **while** al comparar la razón del elemento **k** con el elemento **k-1** y la razón del elemento **k-1** con el elemento **k-2**. Si llama a su secuencia **x**, entonces el código para el enunciado **while** es

```
while abs(x(k)/x(k-1) - x(k-1)/x(k-2))>0.001
```

8.23 Recuerde que en trigonometría la tangente de $\pi/2$ y $-\pi/2$ es infinito. Esto se puede ver a partir del hecho de que

$$\tan(\theta) = \text{sen}(\theta)/\cos(\theta)$$

y puesto que

$$\text{sen}(\pi/2) = 1$$

y

$$\cos(\pi/2) = 0$$

se sigue que

$$\tan(\pi/2) = \text{infinito}$$

Dado que MATLAB usa una aproximación punto flotante de π, calcula la tangente de $\pi/2$ como un número muy grande, pero no infinito.

Conmine al usuario a ingresar un ángulo θ entre $\pi/2$ y $-\pi/2$, inclusive. Si está entre $\pi/2$ y $-\pi/2$, pero no es igual a cualquiera de estos valores, calcule $\tan(\theta)$ y despliegue el resultado en la ventana de comandos. Si es igual a $\pi/2$ o $-\pi/2$, haga el resultado igual a **Inf** y despliegue el resultado en la ventana de comandos. Si está fuera del rango especificado, envíe al usuario un mensaje de error en la ventana de comandos y conmínelo a ingresar otro valor. Continúe conminando al usuario para un nuevo valor de theta hasta que ingrese un número válido.

8.24 Imagine que usted es un orgulloso nuevo padre. Decide iniciar ahora un plan de ahorros universitario para su hijo, con la esperanza de tener suficiente dentro de 18 años para pagar los crecientes costos de una educación. Suponga que sus amigos le dan $1000 para iniciar y que cada mes puede aportar $100. Suponga también que la tasa de interés compuesta mensual es de 6% por año, que equivale a 0.5% cada mes.

Debido a los pagos de interés y a su aportación, cada mes su saldo aumentará en concordancia con la fórmula

Saldo nuevo = saldo anterior + interés + su aportación

Use un bucle **for** para encontrar la cantidad en la cuenta de ahorros cada mes para los siguientes 18 años. (Cree un vector de valores.) Grafique la cantidad en la cuenta como función del tiempo. (Grafique tiempo en el eje horizontal y dólares en el eje vertical.)

8.25 Imagine que tiene una bola de cristal y puede predecir el aumento de porcentaje en matrícula para los siguientes 22 años. El siguiente vector **increase** muestra sus predicciones, en porcentaje para cada año:

```
increase = [10, 8, 10, 16, 15, 4, 6, 7, 8, 10, 8, 12, 14,
  15, 8, 7, 6, 5, 7, 8, 9, 8]
```

Use un bucle **for** para determinar el costo de una educación de cuatro años, si supone que el costo actual para un año en una escuela pública es de $5000.

8.26 Use un enunciado **if** para comparar sus resultados a partir de los dos problemas anteriores. ¿Sus ahorros son suficientes? Envíe un mensaje adecuado a la ventana de comandos.

8.27 **Bucles más rápidos.** Siempre que sea posible, es mejor evitar el uso de bucles **for**, puesto que son lentos para ejecutar.

(a) Genere un vector de 100,000 objetos de dígitos aleatorios llamado **x**; eleve al cuadrado cada elemento en este vector y nombre el resultado **y**; use los comandos **tic** y **toc** para cronometrar la operación.

(b) A continuación, realice la misma operación elemento por elemento en un bucle **for**. Antes de comenzar, limpie los valores en sus variables con

```
clear x y
```

Use **tic** y **toc** para cronometrar la operación.

Dependiendo de cuán rápido corra su computadora, puede requerir detener los cálculos al emitir el comando **ctrl c** en la ventana de comandos.

(c) Ahora convénzase de que suprimir la impresión de respuestas intermedias acelerará la ejecución del código al permitir que estas mismas operaciones corran e impriman las respuestas conforme se calculan. Es casi seguro que necesitará cancelar la ejecución de este bucle debido a la gran cantidad de tiempo que le tomará. ***Recuerde que ctrl c termina el programa***.

(d) Si va a usar un valor constante muchas veces en un bucle **for**, calcúlelo una vez y almacénelo, en lugar de calcularlo cada vez en el bucle. Demuestre el aumento en rapidez de este proceso agregando **(sin(0.3)+cos(pi/3))*5!** a todo valor en el vector largo en un bucle **for**. (Recuerde que ! significa factorial, que se puede calcular con la función MATLAB **factorial**.)

(e) Como se vio en este capítulo, si MATLAB debe aumentar el tamaño de un vector cada vez en un bucle, el proceso tomará más tiempo que si el vector ya tuviese el tamaño adecuado. Demuestre este hecho al repetir la parte (b) de este problema. Cree el siguiente vector de valores **y**, en el que cada elemento sea igual a cero antes de ingresar el bucle **for**:

```
y=zeros(1,100000);
```

Sustituirá los ceros uno a la vez conforme repita los cálculos en el bucle.



CAPÍTULO 9

Álgebra matricial

Objetivos

Después de leer este capítulo, el alumno será capaz de

- realizar las operaciones básicas del álgebra de matrices.
- resolver ecuaciones simultáneas con el uso de las operaciones matriciales MATLAB.
- usar algunas matrices especiales de MATLAB.

INTRODUCCIÓN

Con frecuencia, los términos *arreglo* y *matriz* se usan de manera intercambiable en ingeniería. Sin embargo, técnicamente, un arreglo es un agrupamiento ordenado de información, mientras que una matriz es un arreglo numérico bidimensional que se usa en álgebra lineal. Los arreglos pueden contener información numérica, pero también pueden contener datos carácter, datos simbólicos, etcétera. Por tanto, no todos los arreglos son matrices. Sólo aquéllos sobre los que se tenga intención de realizar transformaciones lineales satisfacen la definición estricta de una matriz.

El álgebra matricial se usa de manera extensa en aplicaciones de ingeniería. Las matemáticas del álgebra matricial se introducen por primera vez en los cursos de álgebra universitaria y se extiende en cursos de álgebra lineal y cursos de ecuaciones diferenciales. Los estudiantes comienzan a usar regularmente el álgebra matricial en clases de estática y dinámica.

9.1 OPERACIONES Y FUNCIONES DE MATRICES

En este capítulo se introducen las funciones y operadores MATLAB que tienen intención específica para usarse en álgebra matricial. Estas funciones y operadores se contrastan con las funciones y operadores de arreglos de MATLAB, de los que difieren significativamente.

9.1.1 Transpuesta

El operador **transpose** (transpuesta) cambia las filas de una matriz en columnas y las columnas en filas. En los textos de matemáticas, con frecuencia verá el transpuesto indicado con el superíndice T (como en A^T). No obstante, no confunda esta notación con la sintaxis MATLAB: en MATLAB, el operador transpuesto es un solo apóstrofe (**'**), de modo que el transpuesto de la matriz **A** es **A'**.

Considere la siguiente matriz y su transpuesta:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ \textcircled{7} & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 4 & \textcircled{7} & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

arreglo: agrupamiento ordenado de información

Las filas y columnas se cambiaron. Note que el valor en la posición (3,1) de A ahora se movió a la posición (1,3) de A^T, y el valor en la posición (4,2) de A ahora se movió a la posición (2,4) de A^T. En general, los subíndices de fila y columna (también llamados números índice) se intercambian para formar la transpuesta.

En MATLAB, uno de los usos más comunes de la operación transponer es cambiar los vectores fila en vectores columna. Por ejemplo:

```
A = [1 2 3];
A'
```

regresa

matriz: arreglo numérico bidimensional que se usa en álgebra lineal

```
A = 1
    2
    3
```

Cuando se usa con números complejos, la operación transponer regresa la conjugada compleja. Por ejemplo, se puede definir un vector de números negativos, sacar la raíz cuadrada y luego transponer la matriz resultante de números complejos. Por ende, el código

```
x = [-1:-1:-3]
```

regresa

Idea clave: los términos arreglo y matriz con frecuencia se usan de manera intercambiable.

```
x =
      -1    -2    -3
```

Entonces, al sacar la raíz cuadrada con el código

```
y=sqrt(x)
y =
  0 + 1.0000i        0 + 1.4142i        0 + 1.7321i
```

transpuesto: cambio de las posiciones de filas y columnas

y finalmente trasponer **y**

```
y'
```

produce

```
ans =
  0 - 1.0000i
  0 - 1.4142i
  0 - 1.7321i
```

Note que los resultados (**y'**) son las conjugadas complejas de los elementos en **y**.

9.1.2 Producto punto

El producto punto (a veces llamado producto escalar) es la suma de los resultados que obtiene cuando multiplica dos vectores, elemento por elemento. Considere los dos vectores siguientes:

```
A = [ 1  2  3];
B = [ 4  5  6];
```

El resultado de la multiplicación arreglo de estos dos vectores es

```
y =A.*B
y =
      4      10      18
```

Si suma los elementos obtiene el producto punto:

```
sum(y)
ans=
      32
```

Un texto de matemáticas representaría el producto punto como

producto punto: suma de los resultados de las multiplicaciones de arreglo de dos vectores

$$\sum_{i=1}^{n} A_i \cdot B_i$$

que se podría escribir en MATLAB como

```
sum(A.*B)
```

MATLAB incluye una función llamada **dot** para calcular el producto punto:

```
dot(A,B)
ans =
      32
```

No importa si **A** y **B** son vectores fila o columna, en tanto tengan el mismo número de elementos.

El producto punto encuentra amplio uso en aplicaciones de ingeniería, tal como se usa al calcular el centro de gravedad en el ejemplo 9.1 y al realizar álgebra vectorial como en el ejemplo 9.2.

Sugerencia

Con los productos punto, no importa si ambos vectores son filas, ambos son columnas o uno es una fila y el otro una columna. Tampoco importa qué orden use para realizar el proceso: el resultado de **dot(A,B)** es el mismo que el de **dot(B,A)**. Esto no ocurre con la mayoría de operaciones matriciales.

EJEMPLO 9.1

Cálculo del centro de gravedad

La masa de un vehículo espacial es una cantidad extremadamente importante. Grupos enteros de personas en el proceso de diseño siguen la pista de la ubicación y masa de cada tuerca y tornillo. No sólo es importante la masa total del vehículo, también la información acerca de la masa se usa para determinar el centro de gravedad del vehículo. Una razón por la que el centro de gravedad es importante es que los cohetes caen si el centro de presión está adelante del centro de gravedad (figura 9.1). Puede demostrar este principio con un avión de papel. Ponga un clip en la punta del avión de papel y observe cómo cambia el patrón de vuelo.

Aunque encontrar el centro de gravedad es un cálculo bastante directo, se vuelve más complicado cuando usted se da cuenta de que tanto la masa del vehículo como la distribución de masa cambian conforme se quema el combustible.

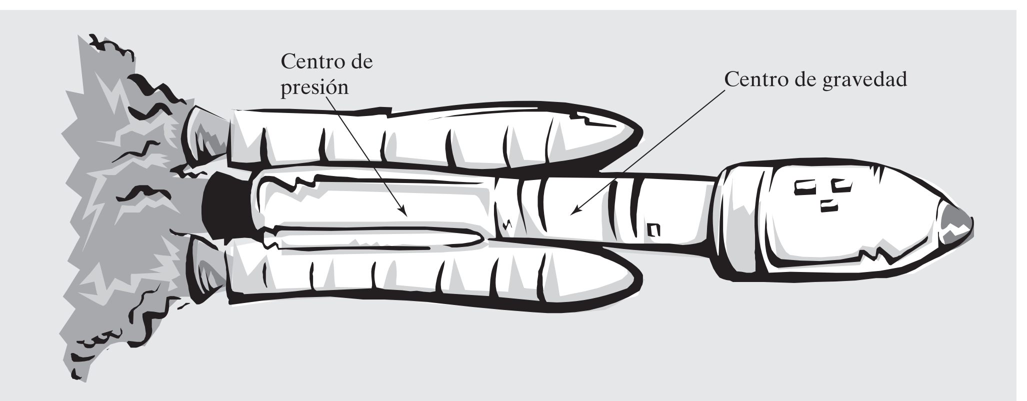


Figura 9.1
El centro de presión necesita estar detrás del centro de gravedad para un vuelo estable.

La ubicación del centro de gravedad se puede calcular al dividir el vehículo en pequeños componentes. En un sistema coordenado rectangular,

$$\bar{x}W = x_1W_1 + x_2W_2 + x_3W_3 + \cdots$$
$$\bar{y}W = y_1W_1 + y_2W_2 + y_3W_3 + \cdots$$
$$\bar{z}W = z_1W_1 + z_2W_2 + z_3W_3 + \cdots$$

donde

$\bar{x}$, $\bar{y}$ y $\bar{z}$ son las coordenadas del centro de gravedad,
W es la masa total del sistema,
$x_1, x_2, x_3,\ldots$, son las coordenadas x de los componentes de sistema 1, 2, 3,..., respectivamente,
$y_1, y_2, y_3,\ldots$, son las coordenadas y de los componentes de sistema 1, 2, 3,..., respectivamente,
$z_1, z_2, z_3,\ldots$, son las coordenadas z de los componentes de sistema 1, 2, 3,..., respectivamente, y
$W_1, W_2, W_3,\ldots$, son los pesos de los componentes de sistema 1, 2, 3,..., respectivamente.

En este ejemplo, se llenará el centro de gravedad de una pequeña colección de los componentes que se usan en un complicado vehículo espacial. (Véase la tabla 9.1.) Este problema se puede formular en términos del producto punto.

1. Establezca el problema.
   Encontrar el centro de gravedad del vehículo espacial.
2. Describa las entradas y salidas.

   Entrada Ubicación de cada componente en un sistema coordenado *x-y-z*
   Masa de cada componente

   Salida Ubicación del centro de gravedad del vehículo

Tabla 9.1 Ubicaciones y masas de componente de vehículo

Objeto	*x*, metros	*y*, metros	*z*, metros	Masa
Tornillo	0.1	2.0	3.0	3.50 gramos
Perno	1.0	1.0	1.0	1.50 gramos
Tuerca	1.5	0.2	0.5	0.79 gramo
Abrazadera	2.0	2.0	4.0	1.75 gramos

Tabla 9.2 Cómo encontrar la coordenada *x* del centro de gravedad

Objeto	*x*, metros		Masa, gramo	*x* × *m*, gramo metros
Tornillo	0.1	×	3.50	= 0.35
Perno	1.0	×	1.50	= 1.50
Tuerca	1.5	×	0.79	= 1.1850
Abrazadera	2.0	×	1.75	= 3.50
Suma			7.54	6.535

3. Desarrolle un ejemplo a mano.
   La coordenada *x* del centro de gravedad es igual a

$$\bar{x} = \frac{\sum_{i=1}^{3} x_i m_i}{m_{\text{Total}}} = \frac{\sum_{i=1}^{3} x_i m_i}{\sum_{i=1}^{3} m_{i1}}$$

   de este modo, a partir de la tabla 9.2,

$$\bar{x} = \frac{6.535}{7.54} = 0.8667 \text{ metro}$$

   Note que la suma de los productos de las coordenadas *x* y las correspondientes masas se podrían expresar como un producto punto.

4. Desarrolle una solución MATLAB.
   El código MATLAB

```
% Ejemplo 9.1
mass = [3.5, 1.5, 0.79, 1.75];
x=[0.1, 1, 1.5, 2];
x_bar=dot(x,mass)/sum(mass)
y=[2, 1, 0.2, 2];
y_bar=dot(y,mass)/sum(mass)
z=[3, 1, 0.5, 4];
z_bar=dot(z,mass)/sum(mass)
```

   regresa el siguiente resultado:

```
x_bar =
  0.8667
y_bar =
  1.6125
z_bar =
  2.5723
```

5. Ponga a prueba la solución.
   Compare la solución MATLAB con la solución a mano. La coordenada *x* parece ser correcta, de modo que las coordenadas *y* y *z* probablemente también sean correctas. Graficar los resultados también le ayudaría a evaluarlos:

```
plot3(x,y,z,'o',x_bar,y_bar,z_bar,'s')
grid on
```

```
xlabel('eje x')
ylabel('eje y')
zlabel('eje z')
title('Centro de gravedad')
axis([0,2,0,2,0,4])
```

La gráfica resultante se muestra en la figura 9.2.

Ahora que se sabe que el programa funciona, se le puede usar para cualquier número de objetos. El programa será el mismo para 3 componentes como para 3000 componentes.

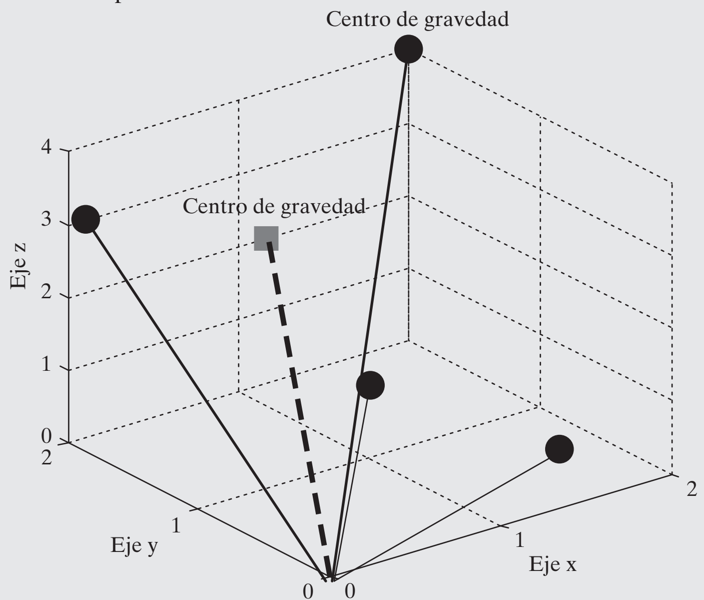


Figura 9.2
Centro de gravedad de algunos datos de muestra. Esta gráfica se mejoró con el uso de las herramientas de graficación interactivas de MATLAB.

EJEMPLO 9.2

Vectores fuerza

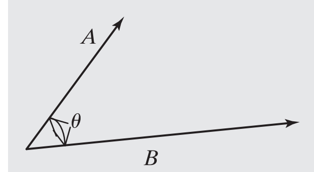


Figura 9.3
Los vectores fuerza se usan en el estudio de estática y dinámica.

La estática es el estudio de las fuerzas en los sistemas que no se mueven (y, por tanto, están *estáticos*). Dichas fuerzas usualmente se describen como vectores. Si suma los vectores, puede determinar la fuerza total sobre un objeto. Considere los dos vectores fuerza **A** y **B** que se muestran en la figura 9.3. Cada uno tiene una magnitud y una dirección. Una notación típica mostraría estos vectores como $\vec{A}$ y $\vec{B}$, pero representaría la magnitud de cada uno (su longitud física) como A y B. Los vectores también se podrían representar en términos de sus magnitudes a lo largo de los ejes x, y y z, multiplicada por un vector unitario $(\vec{i}, \vec{j}, \vec{k})$. Entonces

$$\vec{A} = A_x\vec{i} + A_y\vec{j} + A_z\vec{k}$$

y

$$\vec{B} = B_x\vec{i} + B_y\vec{j} + B_z\vec{k}$$

El producto punto de $\vec{A}$ y $\vec{B}$ es igual a la magnitud de $\vec{A}$ por la magnitud de $\vec{B}$, por el coseno del ángulo entre ellos:

$$\vec{A} \cdot \vec{B} = AB\ \cos(\theta)$$

Encontrar la magnitud de un vector involucra el uso del teorema de Pitágoras. En el caso de tres dimensiones,

$$A = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

Se puede usar MATLAB para resolver problemas como éste si se define el vector $\vec{A}$ como

```
A = [Ax  Ay  Az]
```

donde **Ax**, **Ay** y **Az** son las magnitudes componentes en las direcciones x, y y z, respectivamente. Como problema MATLAB, use el producto punto para encontrar el ángulo entre los siguientes dos vectores fuerza:

$$\vec{A} = 5\vec{i} + 6\vec{j} + 3\vec{k}$$
$$\vec{B} = 1\vec{i} + 3\vec{j} + 2\vec{k}$$

1. Establezca el problema.
   Encontrar el ángulo entre dos vectores fuerza.
2. Describa las entradas y salidas.

   Entrada $\vec{A} = 5\vec{i} + 6\vec{j} + 3\vec{k}$

   $\vec{B} = 1\vec{i} + 3\vec{j} + 2\vec{k}$

   Salida θ, el ángulo entre los dos vectores
3. Desarrolle un ejemplo a mano.

$$\vec{A}\cdot\vec{B} = 5\cdot 1 + 6\cdot 3 + 3\cdot 2 = 29$$
$$A = \sqrt{5^2 + 6^2 + 3^2} = 8.37$$
$$B = \sqrt{1^2 + 3^2 + 2^2} = 3.74$$
$$\cos(\theta) = \vec{A}\cdot\vec{B}/AB = 0.9264$$
$$\cos^{-1}(\theta) = 0.386$$

4. Desarrolle una solución MATLAB.
   El código MATLAB

```
%Ejemplo 9.2
%Encontrar el ángulo entre dos vectores fuerza
%Defina los vectores
A = [5 6 3];
B = [1 3 2];
%Calcule la magnitud de cada vector
mag_A = sqrt(sum(A.^2));
mag_B = sqrt(sum(B.^2));
%Calcule el coseno de theta
cos_theta = dot(A,B)/(mag_A*mag_B);
%Encuentre theta
theta = acos(cos_theta);
%Envíe los resultados a la ventana de comandos
fprintf('El ángulo entre los vectores es %4.3f radianes
  \n',theta)
fprintf('o %6.2f grados \n',theta*180/pi)
```

genera la siguiente interacción en la ventana de comandos:

```
El ángulo entre los vectores es 0.386 radianes
o 22.12 grados
```

5. Ponga a prueba la solución.
   En este caso, sólo se reprodujo la solución a mano en MATLAB. Sin embargo, hacerlo así da la confianza en el proceso de solución, de modo que se podría expandir el problema para permitir al usuario ingresar cualquier par de vectores. Considere este ejemplo.

```
%Ejemplo 9.2 - expandido
%Encontrar el ángulo entre dos vectores fuerza
%Defina los vectores
disp('Las magnitudes de los componentes se deben ingresar')
disp('en notación matricial, i.e.')
disp('[ A B C]')
A = input('Ingrese las magnitudes de los componentes x y z
    del vector A:   ')
B = input('Ingrese las magnitudes de los componentes x y z
    del vector B:   ')
%Calcule la magnitud de cada vector
mag_A = sqrt(sum(A.^2));
mag_B = sqrt(sum(B.^2));
%Calcule el coseno de theta
cos_theta = dot(A,B)/(mag_A*mag_B);
%Encuentre theta
theta = acos(cos_theta);
%Envíe los resultados a la ventana de comandos
fprintf('El ángulo entre los vectores es %4.3f radianes
  \n',theta)
fprintf('o %6.2f grados \n',theta*180/pi)
```

da la siguiente interacción en la ventana de comandos:

```
Las magnitudes de los componentes se deben ingresar
en notación matricial, i.e.
[ A B C]
Ingrese las magnitudes de los componentes x y z del vector
A:   [1 2 3]
A =
     1     2     3
Ingrese las magnitudes de los componentes x y z del vector
B:   [4 5 6]
B =
     4     5     6
El ángulo entre los vectores es 0.226 radianes
o 12.93 grados
```

Ejercicio de práctica 9.1

1. Use la función **dot** para encontrar el producto punto de los siguientes vectores:

$$\vec{A} = [1\ 2\ 3\ 4]$$

$$\vec{B} = [12\ 20\ 15\ 7]$$

2. Encuentre el producto punto de $\vec{A}$ y $\vec{B}$ al sumar los productos arreglo de $\vec{A}$ y $\vec{B}$ (**sum(A.*B)**).
3. Un grupo de amigos fue a un establecimiento local de comida rápida. Ordenaron cuatro hamburguesas a $0.99 cada una, tres refrescos a $1.49 cada uno, una malteada a $2.50, dos órdenes de papas fritas a $0.99 cada una y dos órdenes de anillos de cebolla a $1.29. Use el producto punto para determinar la cuenta.

9.1.3 Multiplicación matricial

La multiplicación matricial es similar al producto punto. Si usted define

```
A = [1 2 3]
B = [ 3;
      4;
      5]
```

entonces

```
A*B
ans =
 26
```

produce el mismo resultado que

```
dot(A,B)
ans =
 26
```

Idea clave: la multiplicación matricial resulta en un arreglo en el que cada elemento es un producto punto.

La multiplicación matricial resulta en un arreglo en el que cada elemento es un producto punto. El ejemplo anterior sólo es el caso más simple. En general, los resultados se encuentran al tomar el producto punto de cada fila en la matriz **A** con cada columna en la matriz **B**. Por ejemplo, si

```
A = [  1 2 3;
       4 5 6  ]
```

y

```
B = [  10   20  30;
       40   50  60;
       70   80  90   ]
```

entonces el primer elemento de la matriz resultante es el producto punto de la fila 1 de la matriz **A** y la columna 1 de la matriz **B**, el segundo elemento es el producto punto de la fila 1 de la matriz **A** y la columna 2 de la matriz **B**, etcétera. Una vez que se encuentra el producto punto para la primera fila de la matriz **A** con todas las columnas de la matriz **B**, se comienza de nuevo con la fila 2 de la matriz **A**. Por ende,

```
C = A*B
```

regresa

```
C =
  300    360    420
  660    810    960
```

Considere el resultado en la fila 2, columna 2, de la matriz **C**. Se puede llamar a este resultado **C(2,2)**. Es el producto punto de la fila 2 de la matriz **A** y la columna 2 de la matriz **B**:

```
dot(A(2,:), B(:,2))
ans =
    810
```

Idea clave: la multiplicación matricial no es conmutativa.

Esta relación se podría expresar en notación matemática (en lugar de sintaxis MATLAB) como

$$C_{i,j} = \sum_{k=1}^{N} A_{i,k} B_{k,j}$$

Puesto que la multiplicación matricial es una serie de productos punto, el número de columnas en la matriz A debe ser igual al número de filas en la matriz B. Si la matriz A es una matriz $m \times n$, la matriz B debe ser $n \times p$, y los resultados serán una matriz $m \times p$. En este ejemplo, A es una matriz 2×3 y B es una matriz 3×3. El resultado es una matriz 2×3.

conmutativa: el orden de la operación no importa

Una forma de visualizar este conjunto de reglas es escribir el tamaño de las dos matrices uno junto al otro, en el orden de su operación. En este ejemplo, se tiene

$$2 \times 3 \quad 3 \times 3$$

Los dos números internos deben coincidir, y los dos números exteriores determinan el tamaño de la matriz resultante.

En general, la multiplicación matricial no es conmutativa, lo que significa que, en MATLAB,

$$\mathbf{A*B} \neq \mathbf{B*A}$$

Se puede ver esto en el ejemplo: cuando se invierte el orden de las matrices, se tiene

$$3 \times 3 \quad 2 \times 3$$

y ya no es posible obtener el producto punto de las filas en la primera matriz y las filas en la segunda matriz. Si ambas matrices son cuadradas, de hecho se puede calcular una respuesta para **A*B** y una respuesta para **B*A**, pero las respuestas no son iguales. Considere este ejemplo:

```
A=[1 2 3
4 5 6
7 8 9];

B=[2 3 4
5 6 7
8 9 10];

A*B

ans =

   36    42    48
   81    96   111
  126   150   174

B*A

ans =

   42    51    60
   78    96   114
  114   141   168
```

EJEMPLO 9.3

Uso de la multiplicación matricial para encontrar el centro de gravedad

En el ejemplo 9.1 se usó el producto punto para encontrar el centro de gravedad de un vehículo espacial. También se podría usar la multiplicación matricial para hacer el cálculo en un paso, en lugar de calcular cada coordenada por separado. En este ejemplo se repite la tabla 9.1 por claridad.

1. Establezca el problema.
   Encontrar el centro de gravedad del vehículo espacial.
2. Describa las entradas y salidas.

   Entrada Ubicación de cada componente en un sistema coordenado *x-y-z*
   Masa de cada componente

   Salida Ubicación del centro de gravedad del vehículo

3. Desarrolle un ejemplo a mano.
   Se puede crear una matriz bidimensional que contenga toda la información acerca de las coordenadas y una correspondiente matriz unidimensional que contenga información acerca de la masa. Si hay n componentes, la información coordenada debe estar en una matriz $3 \times n$ y las masas deben estar en una matriz $n \times 1$. El resultado entonces estaría en una matriz 3×1 que representa las coordenadas *xyz* del centro de gravedad por la masa total.
4. Desarrolle una solución MATLAB.

```
% Ejemplo 9.3
coord =     [0.1         2          3
             1           1          1
             1.5         0.2        0.5
             2           2          4 ]';
mass = [3.5, 1.5, 0.79, 1.75]';
location=coord*mass/sum(mass)
```

   envía los siguientes resultados a la pantalla:

```
location =
  0.8667
  1.6125
  2.5723
```

5. Ponga a prueba la solución.
   Los resultados son los mismos que los del ejemplo 9.1.

Tabla 9.1 Ubicaciones y masas de componente de vehículo

Objeto	*x*, metros	*y*, metros	*z*, metros	Masa
Tornillo	0.1	2.0	3.0	3.50 gramos
Perno	1.0	1.0	1.0	1.50 gramos
Tuerca	1.5	0.2	0.5	0.79 gramo
Abrazadera	2.0	2.0	4.0	1.75 gramos

Ejercicio de práctica 9.2

¿Cuáles de los siguientes conjuntos de matrices se pueden multiplicar entre sí?

1. $A = \begin{bmatrix} 2 & 5 \\ 2 & 9 \\ 6 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 5 \\ 2 & 9 \\ 6 & 5 \end{bmatrix}$

2. $A = \begin{bmatrix} 2 & 5 \\ 2 & 9 \\ 6 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 3 & 12 \\ 5 & 2 & 9 \end{bmatrix}$

3. $A = \begin{bmatrix} 5 & 1 & 9 \\ 7 & 2 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 8 & 5 \\ 4 & 2 \\ 8 & 9 \end{bmatrix}$

4. $A = \begin{bmatrix} 1 & 9 & 8 \\ 8 & 4 & 7 \\ 2 & 5 & 3 \end{bmatrix} \quad B = \begin{bmatrix} 7 \\ 1 \\ 5 \end{bmatrix}$

Demuestre que, para cada caso, $A \cdot B \neq B \cdot A$.

9.1.4 Potencias de matrices

Idea clave: una matriz debe ser cuadrada para elevarla a una potencia.

Elevar una matriz a una potencia es equivalente a multiplicar la matriz por sí misma el número de veces requerido. Por ejemplo, A^2 es lo mismo que $A \cdot A$, A^3 es lo mismo que $A \cdot A \cdot A$. Al recordar que el número de columnas en la primera matriz de una multiplicación debe ser igual al número de filas en la segunda matriz se ve que, con la finalidad de elevar una matriz a una potencia, la matriz debe ser cuadrada (tener el mismo número de filas y columnas). Considere la matriz

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Si se intentara elevar al cuadrado esta matriz, se obtendría un enunciado de error porque las filas y columnas no coinciden:

$$2 \times 3 \qquad 2 \times 3$$

filas y columnas deben coincidir

Sin embargo, considere otro ejemplo. El código

```
A=randn(3)
```

Idea clave: la multiplicación matricial y la multiplicación de matrices son operaciones diferentes y producen resultados diferentes.

crea una matriz 3 × 3 de números aleatorios, tales como

```
A =
   -1.3362   -0.6918   -1.5937
    0.7143    0.8580   -1.4410
    1.6236    1.2540    0.5711
```

> **Sugerencia**
>
> Recuerde que **randn** produce números aleatorios, de modo que su computadora puede producir números diferentes a los que se mencionan aquí.

Si se eleva al cuadrado esta matriz, el resultado también es una matriz 3 × 3:

```
A^2
ans =
   -1.2963   -1.6677    2.2161
   -2.6811   -1.5650   -3.1978
   -0.3463    0.6690   -4.0683
```

Elevar una matriz a una potencia no entera da un resultado complejo:

```
A^1.5
ans =
  -1.8446 - 0.0247i   -1.5333 + 0.0153i   -0.3150 - 0.0255i
  -0.7552 + 0.0283i    0.0668 - 0.0176i   -3.0472 + 0.0292i
   1.3359 + 0.0067i    1.5292 - 0.0042i   -1.5313 + 0.0069i
```

Note que elevar **A** a la **potencia matricial** de dos es diferente de elevar **A** a la **potencia de arreglo** de dos:

```
C = A.^2;
```

Elevar **A** a la potencia de arreglo de dos produce los siguientes resultados:

```
C =
  1.7854    0.4786    2.5399
  0.5102    0.7362    2.0765
  2.6361    1.5725    0.3262
```

9.1.5 Inverso de matriz

En matemáticas, ¿qué se entiende cuando se dice "tomar el inverso"? Para una función, el inverso "deshace" la función o lo lleva de vuelta adonde se comenzó. Por ejemplo, $\text{sen}^{-1}(x)$ es la función inversa de $\text{sen}(x)$. Se puede demostrar la relación en MATLAB:

```
asin(sin(3))
ans =
  3
```

(Recuerde que la sintaxis MATLAB para el seno inverso es asin.)

> **Sugerencia**
>
> Recuerde que $\text{sen}^{-1}(x)$ no significa lo mismo que $1/\text{sen}(x)$. Los textos de matemáticas más actuales usan la notación $\text{sen}^{-1}(x)$, pero en su calculadora y en los programas de cómputo $\text{sen}^{-1}(x)$ se representa como $\text{asen}(x)$.

Otro ejemplo de funciones que son inversas es ln(x) y e^x:

```
log(exp(3))   (Recuerde que la sintaxis MATLAB para el
               logaritmo natural es log, no ln.)
ans =
       3
```

Pero, ¿qué significa tomar el inverso de un número? Una forma de considerarlo es que, si usted operó sobre el número 1 al multiplicarlo por un número, ¿cómo podría deshacer esta operación y obtener el número 1 de vuelta? Obviamente, necesitaría dividir por su número, o multiplicar por 1 sobre el número. Esto conduce a la conclusión de que $1/x$ y x son inversos, pues

Idea clave: una función por su inverso es igual a uno.

$$\frac{1}{x}x = 1$$

Desde luego, éstos son inversos *multiplicativos*, en oposición a la función inversa que se discutió primero. (También hay inversos aditivos, como $-a$ y a.) Finalmente, ¿cuál es el inverso de una matriz? Es la matriz por la que se necesita multiplicar, en álgebra matricial, para obtener la matriz identidad. La matriz identidad consta de unos en la diagonal principal y ceros en todas las otras posiciones:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

La operación inversa es una de las pocas multiplicaciones matriciales conmutativa; esto es,

$$A^{-1}A = AA^{-1} = 1$$

Con la finalidad de que el enunciado anterior sea verdadero, la matriz A debe ser cuadrada, lo que conduce a la conclusión de que, para que una matriz tenga un inverso, debe ser cuadrada.

Estos conceptos se pueden demostrar en MATLAB primero al definir una matriz y luego al experimentar con su comportamiento. La "matriz mágica", en la que la suma de las filas es igual a la suma de las columnas, así como la suma de cada diagonal, es fácil de crear, de modo que se le elegirá para el experimento:

```
A=magic(3)
A =
    8     1     6
    3     5     7
    4     9     2
```

MATLAB ofrece dos enfoques para encontrar el inverso de una matriz. Se podría elevar A a la potencia -1 con el código

```
A^-1
ans =
     0.1472   -0.1444    0.0639
    -0.0611    0.0222    0.1056
    -0.0194    0.1889   -0.1028
```

o se podría usar la función interna **inv**:

```
inv(A)
ans =
   0.1472   -0.1444    0.0639
  -0.0611    0.0222    0.1056
  -0.0194    0.1889   -0.1028
```

Al usar cualquier enfoque, se puede demostrar que multiplicar el inverso de A por A produce la matriz identidad:

```
inv(A)*A
ans =
    1.0000    0         -0.0000
    0         1.0000     0
    0         0.0000     1.0000
```

y

```
A*inv(A)
ans =
    1.0000     0         -0.0000
   -0.0000     1.0000     0
    0.0000     0          1.0000
```

Determinar el inverso de una matriz a mano es difícil, por lo que se dejará dicho ejercicio a un curso de matemáticas matriciales. Existen matrices para las que no existe un inverso; estas matrices se llaman **matrices singulares** o **matrices mal condicionadas**. Cuando usted intenta calcular el inverso de una matriz mal condicionada en MATLAB, se envía un mensaje de error a la ventana de comandos.

matriz singular: matriz que no tiene inverso

La matriz inversa se usa ampliamente en álgebra matricial, aunque rara vez es la forma más eficiente para resolver un problema desde un punto de vista computacional. Esta materia se discute ampliamente en cursos de álgebra lineal.

9.1.6 Determinantes

Los determinantes se usan en álgebra lineal y se relacionan con la matriz inversa. Si el determinante de una matriz es 0, la matriz no tiene inverso y se dice que es singular. Los determinantes se calculan al multiplicar los elementos a lo largo de las diagonales izquierda a derecha de la matriz y restar el producto de las diagonales derecha a izquierda. Por ejemplo, para una matriz 2×2

Idea clave: si el determinante es cero, la matriz no tiene inverso.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

el determinante es

$$|A| = A_{11}A_{22} - A_{12}A_{21}$$

Por tanto, para

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$|A| = (1)(4) - (2)(3) = -2$$

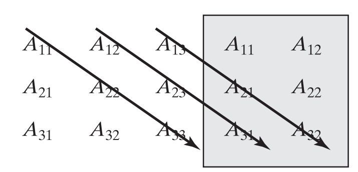
MATLAB tiene una función determinante interna, **det**, que encontrará el determinante por usted:

```
A=[1 2;3 4];
det(A)
ans =
  -2
```

Imaginar las diagonales para una matriz 3×3

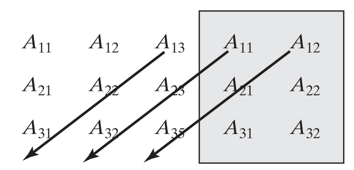
$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

es un poco más difícil. Si copia las primeras dos columnas de la matriz en las columnas 4 y 5 se vuelve más fácil de ver. Multiplique cada diagonal izquierda a derecha y súmelas:



$$(A_{11}A_{22}A_{33}) + (A_{12}A_{23}A_{31}) + (A_{13}A_{21}A_{32})$$

Luego multiplique cada diagonal derecha a izquierda y súmelas:



$$(A_{13}A_{22}A_{31}) + (A_{11}A_{23}A_{32}) + (A_{12}A_{21}A_{33})$$

Finalmente, reste el segundo cálculo del primero. Por ejemplo, puede tener

$$|A| = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = (1 \times 5 \times 9) + (2 \times 6 \times 7) + (3 \times 4 \times 8)$$

$$- (3 \times 5 \times 7) - (1 \times 6 \times 8) - (2 \times 4 \times 9) = 225 - 225 = 0$$

Al usar MATLAB para el mismo cálculo se produce

```
A=[1 2 3;4 5 6;7 8 9];
det(A)
ans =
  0
```

Puesto que se sabe que las matrices con un determinante cero no tienen inversos, vea lo que ocurre cuando se pide a MATLAB encontrar el inverso de A*:

```
inv(A)
Warning: La matriz está cerca de ser singular o está mal escalada.
     Los resultados pueden ser imprecisos. RCOND = 1.541976e-018.
ans =
 1.0e+016 *
 -0.4504    0.9007   -0.4504
```

* El mensaje de advertencia dice: la matriz está cerca de ser singular o está mal escalada. Los resultados pueden ser imprecisos.

Ejercicio de práctica 9.3

1. Encuentre el inverso de las siguientes matrices mágicas, tanto con la función **inv** como al elevar la matriz a la potencia −1:

   a. magic(3)
   b. magic(4)
   c. magic(5)

2. Encuentre el determinante de cada una de las matrices de la parte 1.
3. Considere la siguiente matriz:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

   ¿Esperaría que fuera singular o no? (Recuerde que las matrices singulares tienen un determinante 0 y no tienen inverso.)

```
  0.9007   -1.8014    0.9007
 -0.4504    0.9007   -0.4504
```

9.1.7 Productos cruz

Los productos cruz a veces se llaman productos vectoriales porque, a diferencia de los productos punto, que regresan un escalar, el resultado de un producto cruz es un vector. El vector resultante siempre está en ángulos rectos (normal) al plano definido por los dos vectores entrada, una propiedad que se llama *ortogonalidad*.

Idea clave: el resultado de un producto cruz es un vector.

ortogonal: en ángulos rectos

Considere dos vectores en el espacio que representen tanto una dirección como una magnitud. (La fuerza usualmente se representa de esta forma.) Matemáticamente,

$$\vec{A} = A_x\vec{i} + A_y\vec{j} + A_z\vec{k}$$
$$\vec{B} = B_x\vec{i} + B_y\vec{j} + B_z\vec{k}$$

Los valores A_x, A_y, A_z y B_x, B_y, B_z representan la magnitud del vector en las direcciones x, y y z, respectivamente. Los símbolos $\vec{i}, \vec{j}, \vec{k}$ representan vectores unitarios en las direcciones x, y y z. El *producto cruz* de $\vec{A}$ y $\vec{B}$, $\vec{A} \times \vec{B}$, se define como

$$\vec{A} \times \vec{B} = (A_yB_z - A_zB_y)\vec{i} + (A_zB_x - A_xB_z)\vec{j} + (A_xB_y - A_yB_x)\vec{k}$$

Puede ver esta operación al crear una tabla

$$\begin{matrix} i & j & k \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{matrix}$$

y luego repetir las dos primeras columnas al final de la tabla:

$$\begin{matrix} i & j & k & i & j \\ A_x & A_y & A_z & A_x & A_y \\ B_x & B_y & B_z & B_x & B_y \end{matrix}$$

El componente del producto cruz en la dirección *i* se encuentra al obtener el producto A_yB_z y restarle el producto A_zB_y:

$$\begin{array}{ccccc} \textcircled{i} & j & k & i & j \\ A_x & A_y & A_z & A_x & A_y \\ B_x & B_y & B_z & B_x & B_y \end{array}$$

Al moverse a través del diagrama, el componente del producto cruz en la dirección *j* se encuentra al obtener el producto A_zB_x y restarle el producto A_xB_z:

$$\begin{array}{ccccc} i & \textcircled{j} & k & i & j \\ A_x & A_y & A_z & A_x & A_y \\ B_x & B_y & B_z & B_x & B_y \end{array}$$

Por último, el componente del producto cruz en la dirección *k* se encuentra al obtener el producto A_xB_y y restarle el producto A_yB_x:

$$\begin{array}{ccccc} i & j & \textcircled{k} & i & j \\ A_x & A_y & A_z & A_x & A_y \\ B_x & B_y & B_z & B_x & B_y \end{array}$$

Sugerencia

Es posible que haya notado que el producto cruz sólo es un caso especial de un determinante cuya primera fila se compone de vectores unitarios.

En MATLAB, el producto cruz se encuentra al usar la función **cross**, que requiere dos entradas: los vectores **A** y **B**. Cada uno de estos vectores MATLAB debe tener tres elementos, pues representan los componentes vectoriales en el espacio. Por ejemplo, se puede tener

```
A=[1 2 3];
```
(que representa $\vec{A} = 1\vec{i} + 2\vec{j} + 3\vec{k}$)

```
B=[4 5 6];
```
(que representa $\vec{B} = 4\vec{i} + 5\vec{j} + 6\vec{k}$)

```
cross(A,B)
ans =
   -3     6     -3
```
(que representa $\vec{C} = -3\vec{i} + 6\vec{j} - 3\vec{k}$)

Considere dos vectores en el plano *x-y* (sin componente *z*):

```
A=[1 2 0]
B=[3 4 0]
```

La magnitud de estos vectores en la dirección *z* se necesita especificar como cero en MATLAB.

El resultado del producto cruz debe estar en ángulos rectos al plano que contiene los vectores **A** y **B**, lo que dice que en este caso debe estar afuera del plano *x-y*, con sólo un componente *z*.

```
cross(A,B)
ans =
   0     0    -2
```

Los productos cruz tienen amplio uso en estática, dinámica, mecánica de fluidos y problemas de ingeniería eléctrica.

EJEMPLO 9.4

Momento de una fuerza en torno a un punto

El momento de una fuerza en relación con un punto se encuentra al calcular el producto cruz de un vector que define la *posición* de la fuerza con respecto al punto, con el vector fuerza:

$$M_0 = r \times F$$

Considere la fuerza aplicada en el extremo de una palanca, como se muestra en la figura 9.4. Si se aplica una fuerza a la palanca cerca del punto pivote, el efecto es diferente que si se aplica una fuerza más alejada sobre la palanca. Dicho efecto se llama *momento*.

Calcule el momento en torno al punto pivote de una palanca para una fuerza descrita como el vector

$$\vec{F} = -100\vec{i} + 20\vec{j} + 0\vec{k}$$

Suponga que la palanca tiene 12 pulgadas de largo, a un ángulo de 45 grados desde la horizontal. Esto significa que el vector posición se puede representar como

$$\vec{r} = \frac{12}{\sqrt{2}}\vec{i} + \frac{12}{\sqrt{2}}\vec{j} + 0\vec{k}$$

1. Establezca el problema.
   Encontrar el momento de un vector fuerza en torno al punto pivote de una palanca.

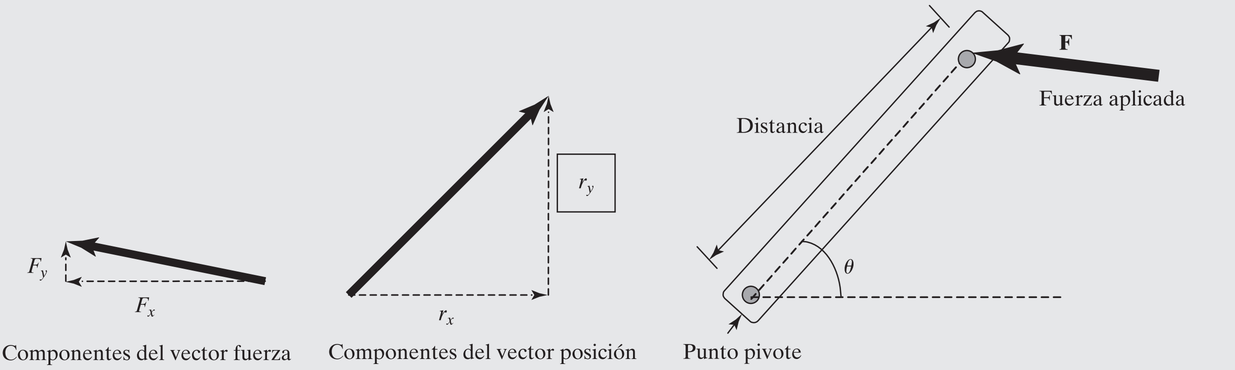


Figura 9.4
La fuerza aplicada a una palanca crea un momento en torno al punto pivote.

2. Describa las entradas y salidas.

Entrada

vector posición $\vec{r} = \frac{12}{\sqrt{2}}\vec{i} + \frac{12}{\sqrt{2}}\vec{j} + 0\vec{k}$

vector fuerza $\vec{F} = -100\vec{i} + 20\vec{j} + 0\vec{k}$

Salida Momento en torno al punto pivote de la palanca

3. Desarrolle un ejemplo a mano.
Visualice el problema como el determinante de un arreglo 3 × 3:

$$M_0 = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ \frac{12}{\sqrt{2}} & \frac{12}{\sqrt{2}} & 0 \\ -100 & 20 & 0 \end{vmatrix}$$

Obviamente, no puede haber componentes $\vec{i}$ o $\vec{j}$ en la respuesta. El momento debe ser

$$M_0 = \left(\frac{12}{\sqrt{2}} \times 20 - \frac{12}{\sqrt{2}} \times (-100)\right) \times \vec{k} = 1018.23\vec{k}$$

4. Desarrolle una solución MATLAB.
El código MATLAB

```
%Ejemplo 9.4
%Momento en torno a un punto pivote
%Defina el vector posición
r = [12/sqrt(2), 12/sqrt(2), 0];
%Defina el vector fuerza
F = [-100, 20, 0];
%Calcule el momento
moment=cross(r,F)
```

regresa el siguiente resultado:

```
moment =
     0      0     1018.23
```

Esto corresponde a un vector momento

$$M_0 = 0\vec{i} + 0\vec{j} + 1018.23\vec{k}$$

Note que el momento está en ángulos rectos al plano definido por los vectores posición y fuerza.

5. Ponga a prueba la solución.
Claramente, las soluciones a mano y MATLAB concuerdan, lo que significa que ahora se puede expandir el programa a una solución más general. Por ejemplo, el siguiente programa solicita al usuario los componentes x, y y z de los vectores posición y fuerza y luego calcula el momento:

```
%Ejemplo 9.4
%Momento en torno a un punto pivote
%Defina el vector posición
  clear,clc
  rx=input('Ingrese el componente x del vector posición: ');
  ry=input('Ingrese el componente y del vector posición: ');
  rz=input('Ingrese el componente z del vector posición: ');
  r = [rx, ry, rz];
  disp('El vector posición es')
  fprintf('%8.2f i + %8.2f j + %8.2f k ft\n',r)
%Defina el vector fuerza
  Fx=input('Ingrese el componente x del vector fuerza: ');
  Fy=input('Ingrese el componente y del vector fuerza: ');
  Fz=input('Ingrese el componente z del vector fuerza: ');
  F = [Fx, Fy, Fz];
  disp('El vector fuerza es')
  fprintf('%8.2f i + %8.2f j + %8.2f k lbf\n',F)
%Calcule el momento
  moment=cross(r,F);
  fprintf('El vector momento en torno al punto pivote es \n')
  fprintf('%8.2f i + %8.2f j + %8.2f k ft-lbf\n',moment)
```

Un ejemplo de interacción en la ventana de comandos es

```
Ingrese el componente x del vector posición: 2
Ingrese el componente y del vector posición: 3
Ingrese el componente z del vector posición: 4
El vector posición es
  2.00 i +     3.00 j  +     4.00 k ft
Ingrese el componente x del vector fuerza: 20
Ingrese el componente y del vector fuerza: 10
Ingrese el componente z del vector fuerza: 30
El vector fuerza es
  20.00 i +    10.00 j  +    30.00 k lbf
El vector momento en torno al punto pivote es
  50.00 i +    20.00 j  +   -40.00 k ft-lbf
```

9.2 SOLUCIONES DE SISTEMAS DE ECUACIONES LINEALES

Considere el siguiente sistema de tres ecuaciones con tres incógnitas:

$$\begin{array}{rrrcr} 3x & +2y & -z & = & 10 \\ -x & +3y & +2z & = & 5 \\ x & -y & -z & = & -1 \end{array}$$

Este sistema de ecuaciones se puede rescribir con las siguientes matrices:

$$A = \begin{bmatrix} 3 & 2 & -1 \\ -1 & 3 & 2 \\ 1 & -1 & -1 \end{bmatrix} \qquad X = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad B = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

Al usar multiplicación matricial se puede escribir entonces el **sistema de ecuaciones**

$$AX = B.$$

9.2.1 Solución con el uso de la matriz inversa

Probablemente la forma más directa de resolver este sistema de ecuaciones es usar la matriz inversa. Dado que se sabe que

$$A^{-1}A = 1$$

se pueden multiplicar ambos lados de la ecuación matricial $AX = B$ por A^{-1} para obtener

$$A^{-1}AX = A^{-1}B$$

lo que produce

$$X = A^{-1}B$$

Como en todas las matemáticas matriciales, el orden de multiplicación es importante. Dado que A es una matriz 3×3, su inverso A^{-1} también es una matriz 3×3. La multiplicación $A^{-1}B$

$$3 \times 3 \qquad 3 \times 1$$

funciona porque las dimensiones coinciden. El resultado es la matriz 3×1 X. Si se cambia el orden a BA^{-1}, las dimensiones ya no coincidirían y la operación sería imposible.

Puesto que en MATLAB la matriz inversa se calcula con la función **inv**, se puede usar el siguiente conjunto de comandos para resolver este problema:

```
A=[3  2  -1; -1  3  2; 1  -1  -1];
B=[10; 5; -1];
X = inv(A)*B
```

Este código regresa

```
X =
  -2.0000
   5.0000
  -6.0000
```

De manera alternativa, podría representar la matriz inversa como **A^-1**, de modo que

```
X=A^-1*B
```

lo que regresa

```
X =
  -2.0000
   5.0000
  -6.0000
```

Aunque esta técnica corresponde bien con el enfoque que se considera en las clases de álgebra cuando se introducen las matrices, no es muy eficiente y puede resultar en excesivos errores de redondeo. En general, se debe evitar usar la matriz inversa para resolver sistemas lineales de ecuaciones.

EJEMPLO 9.5

Resolución de ecuaciones simultáneas: un circuito eléctrico*

Al resolver un problema de circuito eléctrico, uno se encuentra rápidamente empantanado en una gran cantidad de ecuaciones simultáneas. Por ejemplo, considere el circuito eléctrico que se muestra en la figura 9.5. Contiene una sola fuente de voltaje y cinco reóstatos. Puede analizar este circuito al dividirlo en partes más pequeñas y usar dos hechos básicos en torno a la electricidad:

$$\sum \textit{voltaje} \text{ alrededor de un circuito debe ser cero}$$

$$\text{Voltaje} = \text{corriente} \times \text{resistencia } (V = iR)$$

Seguir el lazo inferior izquierdo resulta en la primera ecuación:

$$-V_1 + R_2(i_1 - i_2) + R_4(i_1 - i_3) = 0$$

Seguir el lazo superior resulta en la segunda ecuación:

$$R_1 i_2 + R_3(i_2 - i_3) + R_2(i_2 - i_1) = 0$$

Finalmente, seguir el lazo inferior derecho resulta en la última ecuación:

$$R_3(i_3 - i_2) + R_5 i_3 + R_4(i_3 - i_1) = 0$$

Dado que se conocen todas las resistencias (los valores R) y el voltaje, se tienen tres ecuaciones y tres incógnitas. Ahora se necesita reordenar las ecuaciones de modo que estén en una forma en la que se pueda aplicar una solución matricial. En otras palabras, se necesita aislar las *i*es del modo siguiente:

$$(R_2 + R_4)i_1 + (-R_2)i_2 + (-R_4)i_3 = V_1$$
$$(-R_2)i_1 + (R_1 + R_2 + R_3)i_2 + (-R_3)i_3 = 0$$
$$(-R_4)i_1 + (-R_3)i_2 + (R_3 + R_4 + R_5)i_3 = 0$$

Cree un programa MATLAB para resolver estas ecuaciones con el método de matriz inversa. Permita al usuario ingresar cinco valores de R y el voltaje desde el teclado.

1. Establezca el problema.
   Encontrar las tres corrientes para el circuito que se muestra.

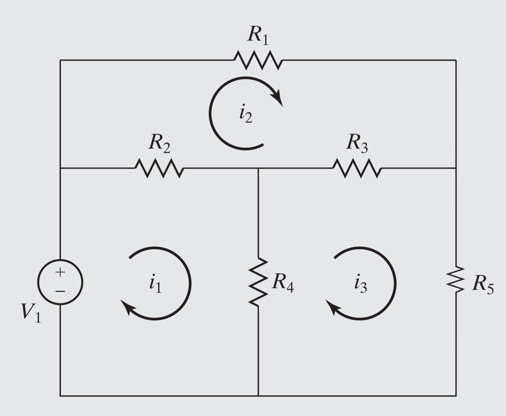


Figura 9.5
Un circuito eléctrico.

*Tomado de *Introduction to MATLAB 7*, de Etter, Kuncicky y Moore (Upper Saddle River, NJ: Pearson Prentice Hall, 2005).

2. Describa las entradas y salidas.

   Entrada Cinco resistencias R_1, R_2, R_3, R_4, R_5 y el voltaje V, proporcionados desde el teclado

   Salida Tres valores de corriente i_1, i_2, i_3

3. Desarrolle una solución a mano.
   Si no hay voltaje aplicado en el circuito, puede no haber corriente, de modo que si se ingresa algún valor para las resistencias y se ingresa cero para el voltaje, la respuesta debe ser cero.

4. Desarrolle una solución MATLAB.
   El código MATLAB

```
%Ejemplo 9.5
%Encontrar corrientes
clear,clc
R1 = input('Ingrese el valor de R1: ');
R2 = input('Ingrese el valor de R2: ');
R3 = input('Ingrese el valor de R3: ');
R4 = input('Ingrese el valor de R4: ');
R5 = input('Ingrese el valor de R5: ');
V = input('Ingrese el valor del voltaje, V: ');
coef = [(R2+R4), -R2, -R4;
     -R2, (R1 + R2 +R3), (-R3);
     -R4, - R3,(R3 + R4 + R5)];
result = [V; 0; 0];
I = inv(coef)*result
```

   genera la siguiente interacción en la ventana de comandos:

```
Ingrese el valor de R1:  5
Ingrese el valor de R2:  5
Ingrese el valor de R3:  5
Ingrese el valor de R4:  5
Ingrese el valor de R5:  5
Ingrese el valor del voltaje, V:  0

I =
       0
       0
       0
```

5. Ponga a prueba la solución.
   A propósito se eligió ingresar un voltaje de cero con la finalidad de comprobar la solución. Los circuitos sin fuerza impulsora (voltaje) no pueden tener un flujo de corriente a través de ellos. Ahora intente el programa con otros valores:

```
Ingrese el valor de R1:  2
Ingrese el valor de R2:  4
Ingrese el valor de R3:  6
Ingrese el valor de R4:  8
Ingrese el valor de R5:  10
Ingrese el valor del voltaje, V:  10
```

   En conjunto, estos valores producen

```
I =
     1.69
     0.97
     0.81
```

9.2.2 Solución con división izquierda de matriz

Una mejor forma de resolver un sistema de ecuaciones lineales es usar una técnica llamada *eliminación gaussiana*. En realidad ésta es la forma en que usted probablemente aprendió a resolver sistemas de ecuaciones en el álgebra de bachillerato. Considere el problema de tres ecuaciones en x, y y z:

Idea clave: la eliminación gaussiana es más eficiente y menos susceptible a error de redondeo que el método de matriz inversa.

$$\begin{array}{rrrcr} 3x & +2y & -z & = & 10 \\ -x & +3y & +2z & = & 5 \\ x & -y & -z & = & -1 \end{array}$$

Para resolver este problema a mano, se considerarían primero las dos primeras ecuaciones en el conjunto y se eliminaría una de las variables, por ejemplo, x. Para hacer esto necesitará multiplicar la segunda ecuación por 3 y luego sumar la ecuación resultante a la primera:

$$\begin{array}{rrrcr} 3x & +2y & -z & = & 10 \\ -3x & +9y & +6z & = & 15 \\ \hline 0 & +11y & +5z & = & 25 \end{array}$$

Ahora se necesita repetir el proceso para la segunda y tercera ecuaciones:

$$\begin{array}{rrrcr} -x & +3y & +2z & = & 5 \\ x & -y & -z & = & -1 \\ \hline 0 & +2y & +z & = & 4 \end{array}$$

En este punto, se eliminó una variable y el problema se redujo a dos ecuaciones y dos incógnitas:

$$\begin{array}{rrcr} 11y & +5z & = & 25 \\ 2y & +z & = & 4 \end{array}$$

Ahora se puede repetir el proceso de eliminación al multiplicar la fila 3 por $-11/2$:

$$\begin{array}{rrcr} 11y & +5z & = & 25 \\ -\frac{11}{2}*2y & -\frac{11}{2}z & = & -\frac{11}{2}*4 \\ \hline 0 & -\frac{1}{2}z & = & 3 \end{array}$$

Finalmente, se puede resolver para z:

$$z = -6$$

Una vez que se conoce el valor de z, se puede sustituir de vuelta en cualquiera de las dos ecuaciones con sólo z y y, a saber,

$$\begin{array}{rrcr} 11y & +5z & = & 25 \\ 2y & +z & = & 4 \end{array}$$

para encontrar que

$$y = 5$$

El último paso es sustituir de nuevo en una de las cuatro ecuaciones originales,

$$\begin{array}{rrrcr} 3x & +2y & -z & = & 10 \\ -x & +3y & +2z & = & 5 \\ x & -y & -z & = & -1 \end{array}$$

para encontrar que

$$x = -2$$

eliminación gaussiana: enfoque organizado para eliminar variables y resolver un conjunto de ecuaciones simultáneas

La técnica de eliminación gaussiana es un enfoque organizado para eliminar variables hasta que sólo existe una incógnita y luego sustituir de nuevo hasta que se determinan todas las incógnitas. En MATLAB se puede usar división izquierda para resolver el problema por eliminación gaussiana. En consecuencia,

```
X = A\B
```

regresa

```
X =
  -2.0000
   5.0000
  -6.0000
```

Obviamente, éste es el mismo resultado que se obtuvo con la solución a mano y el enfoque de matriz inversa. En un problema simple como éste, el error de redondeo y el tiempo de ejecución no son grandes factores para determinar cuál enfoque usar. Sin embargo, algunas técnicas numéricas requieren la solución de matrices con miles o incluso millones de elementos. Los tiempos de ejecución se miden en horas o días para estos problemas, y el error de redondeo y el tiempo de ejecución se convierten en consideraciones cruciales.

No todos los sistemas de ecuaciones lineales tienen una solución única. Si existen menos ecuaciones que variables, el problema está subespecificado. Si hay más ecuaciones que variables, el problema está sobreespecificado. MATLAB incluye funciones que le permitirán resolver cada uno de estos sistemas de ecuaciones al usar enfoques numéricos de mejor ajuste o agregar restricciones. Consulte la función **help** de MATLAB para más información acerca de estas técnicas.

EJEMPLO 9.6

Balance de materia en una unidad de desalinización: resolución de ecuaciones simultáneas

El agua fresca es un recurso escaso en muchas partes del mundo. Por ejemplo, Israel soporta una moderna sociedad industrial en medio de un desierto. Para complementar los recursos acuíferos locales, Israel depende de las plantas de desalinización de agua a lo largo de la costa mediterránea. Estimaciones actuales predicen que la demanda por agua fresca en Israel aumentará 60% hacia el año 2020, y la mayoría de esta nueva agua tendrá que venir de desalinización. Las modernas plantas desalinizadoras usan ósmosis inversa, ¡el proceso que se usa en diálisis renal! Los ingenieros químicos usan ampliamente cálculos de balance de materias para diseñar y analizar plantas como las desalinizadoras de agua en Israel.

Considere la hipotética unidad de desalinización que se muestra en la figura 9.6. El agua salada que fluye en la unidad contiene 4 %w de sal y 96 %w de agua. Dentro de la uni-

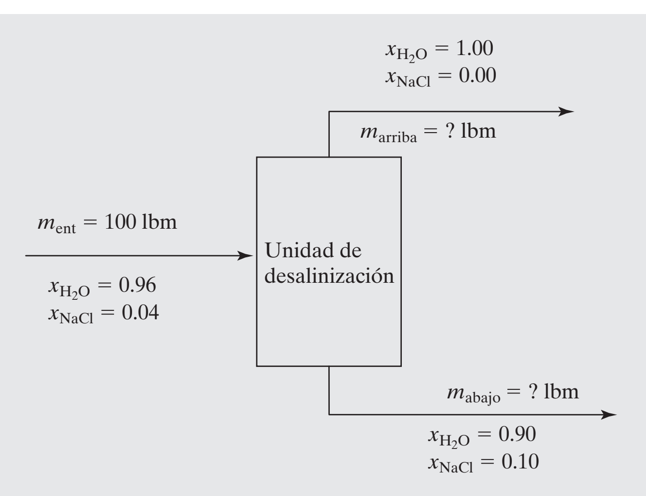


Figura 9.6
La desalinización de agua es una importante fuente de agua fresca para naciones desérticas como Israel.

dad, el agua se separa en dos corrientes mediante una serie de operaciones de ósmosis inversa. La corriente que fluye por arriba casi es agua pura. La restante solución concentrada de agua salada es 10 %w de sal y 90 %w de agua. Calcule las tasas de flujo de masa que vienen de arriba y abajo de la unidad de desalinización.

Este problema requiere la realización de un balance de materias en el reactor tanto para sal como para agua. La cantidad de cualquier componente que fluye en el reactor debe ser el mismo que la cantidad de dicho componente que fluye en las dos corrientes de salida. Esto es,

$$m_{\text{entA}} = m_{\text{arribaA}} + m_{\text{abajoA}}$$

que se podría reescribir como

$$x_A m_{\text{ent total}} = x_{\text{Aarriba}} m_{\text{arriba}} + x_{\text{Aabajo}} m_{\text{abajo}}$$

Por ende, se puede formular este problema como un sistema de dos ecuaciones con dos incógnitas:

$$0.96 \times 100 = 1.00 m_{\text{arriba}} + 0.90 m_{\text{abajo}} \quad \text{(para agua)}$$

$$0.04 \times 100 = 0.00 m_{\text{arriba}} + 0.10 m_{\text{abajo}} \quad \text{(para sal)}$$

1. Establezca el problema.
   Encontrar la masa de agua fresca que se produce y la masa de salmuera eliminada de la unidad de desalinización.
2. Describa las entradas y salidas.

   Entrada Masa de 100 lb en el sistema
   Concentraciones (fracciones de masa) de la corriente de entrada:

$$x_{H^2O} = 0.96$$

$$x_{NaCl} = 0.04$$

Concentraciones (fracciones de masa) en las corrientes de salida:
corriente rica en agua (arriba)

$x_{H^2O} = 1.00$

salmuera (abajo)

$x_{H^2O} = 0.90$

$x_{NaCl} = 0.10$

Salida Masa de salida de la corriente rica en agua (arriba)
Masa de salida de la salmuera (abajo)

3. Desarrolle un ejemplo a mano.
   Dado que la sal (NaCl) está presente sólo en una de las corrientes de salida, es fácil resolver el siguiente sistema de ecuaciones:

$$(0.96)(100) = 1.00m_{arriba} + 0.90m_{abajo} \quad \text{(para agua)}$$

$$(0.04)(100) = 0.00m_{arriba} + 0.10m_{abajo} \quad \text{(para sal)}$$

   Si comienza con el balance de materia sal, se encuentra que

$$4 = 0.1m_{abajo}$$

$$m_{abajo} = 40 \text{ lbm}$$

   Una vez conocido el valor de m_{abajo}, se puede sustituir de nuevo en el balanceo de agua:

$$96 = 1m_{arriba} + (0.90)(40)$$

$$m_{arriba} = 60 \text{ lb}$$

4. Desarrolle una solución MATLAB.
   Se puede usar matemáticas matriciales para resolver este problema una vez se dé cuenta de que es de la forma

$$AX = B$$

   donde A es la matriz coeficiente y, por tanto, las fracciones de masa del agua y la sal. La matriz resultado, B, consiste en la tasa de flujo de masa en el sistema de agua y sal:

$$A = \begin{bmatrix} 1 & 0.9 \\ 0 & 0.1 \end{bmatrix} \qquad B = \begin{bmatrix} 96 \\ 4 \end{bmatrix}$$

   La matriz de incógnitas, X, consiste en las tasas de flujo de masa totales que salen de arriba y abajo de la unidad de desalinización. El uso de MATLAB para resolver este sistema de ecuaciones requiere sólo tres líneas de código:

```
A = [1, 0.9; 0, 0.1];
B = [96; 4];
X = A\B
```

   Este código regresa

```
X =
  60
  40
```

5. Ponga a prueba la solución.
   Note que en este ejemplo se eligió usar división izquierda matricial. Usar el enfoque de matriz inversa produce el mismo resultado:

```
X=inv(A)*B
X =
  60
  40
```

Los resultados de ambos enfoques coinciden con el del ejemplo a mano, pero se puede hacer una comprobación adicional para verificar los resultados. El balance de materia se realizó con base en agua y sal, pero se puede realizar un balanceo adicional sobre la masa *total* que entra y sale del sistema:

$$m_{\text{ent}} = m_{\text{arriba}} + m_{\text{abajo}}$$

$$m_{\text{ent}} = 40 + 60 = 100$$

Verificar que realmente salen del sistema 100 lbm sirve como una confirmación más de que se realizaron los cálculos correctamente.

Aunque fue sencillo resolver a mano el sistema de ecuaciones de este problema, la mayoría de los cálculos de balance de materias reales incluyen más corrientes de proceso y más componentes. Las soluciones matriciales como la que se creó son una importante herramienta para los ingenieros de procesos químicos.

9.3 MATRICES ESPECIALES

MATLAB contiene un grupo de funciones que generan matrices especiales, algunas de las cuales se discuten en esta sección.

9.3.1 Unos y ceros

Las funciones **ones** y **zeros** crean matrices que consisten por completo en unos y ceros, respectivamente. Cuando se usa una sola entrada, el resultado es una matriz cuadrada. Cuando se usan dos entradas, especifican el número de filas y columnas. Por ejemplo,

```
ones(3)
```

regresa

```
ans =
       1     1     1
       1     1     1
       1     1     1
```

y

```
zeros(2,3)
```

regresa

```
ans =
       0     0     0
       0     0     0
```

Si en cualquier función se especifican más de dos entradas, MATLAB crea una matriz multidimensional. Por ejemplo,

```
ones(2,3,2)
ans(:,:,1) =
      1.00        1.00        1.00
      1.00        1.00        1.00
ans(:,:,2) =
      1.00        1.00        1.00
      1.00        1.00        1.00
```

crea una matriz tridimensional con dos filas, tres columnas y dos páginas.

9.3.2 Matriz identidad

Una matriz identidad es una matriz con unos en la diagonal principal y ceros en todas las demás ubicaciones. Por ejemplo, la siguiente matriz es una matriz identidad con cuatro filas y cuatro columnas:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note que la diagonal principal es la diagonal que contiene los elementos en los que el número de fila es el mismo que el número de columna. Los subíndices para los elementos en la diagonal principal son (1,1), (2,2), (3,3), etcétera.

En MATLAB, las matrices identidad se pueden generar con la función **eye**. Los argumentos de la función **eye** son similares a los de las funciones **zeros** y **ones**. Si el argumento de la función es un escalar, como en **eye**(6), la función generará una matriz cuadrada, usando el argumento tanto como el número de filas como el de columnas. Si la función tiene dos argumentos escalares, como en **eye(m,n)**, la función generará una matriz con *m* filas y *n* columnas. Para generar una matriz identidad que sea del mismo tamaño que otra matriz, use la función **size** para determinar el número correcto de filas y columnas. Aunque la mayoría de las aplicaciones usan una matriz identidad cuadrada, la definición se puede extender a matrices no cuadradas. Los siguientes enunciados ilustran estos casos:

```
A = eye(3)
A =
      1     0     0
      0     1     0
      0     0     1
B = eye(3,2)
B =
      1     0
      0     1
      0     0
C = [1, 2, 3 ; 4, 2, 5]
C =
      1     2     3
      4     2     5
D = eye(size(C))
D =
      1     0     0
      0     1     0
```

Sugerencia

Se recomienda que no nombre una matriz identidad **i**, porque **i** ya no representará $\sqrt{-1}$ en algún enunciado que siga.

Recuerde que **A * inv(A)** es igual a la matriz identidad. Esto se puede ilustrar con los siguientes enunciados:

```
A=[1,0,2; -1, 4, -2; 5,2,1]
A =
      1      0     2
     -1      4    -2
      5      2     1
inv(A)
ans =
       -0.2222      -0.1111       0.2222
        0.2500       0.2500       0.0000
        0.6111       0.0556      -0.1111
A*inv(A)
ans =
        1.0000       0            0.0000
       -0.0000       1.0000       0.0000
       -0.0000      -0.0000       1.0000
```

Como se discutió anteriormente, la multiplicación matricial en general no es conmutativa; esto es,

$$AB \neq BA$$

Sin embargo, para matrices identidad

$$AI = IA$$

que se puede demostrar con el siguiente código MATLAB:

```
I = eye(3)
I =
    1     0     0
    0     1     0
    0     0     1
A*I
ans =
    1     0     2
   -1     4    -2
    5     2     1
I*A
ans =
    1     0     2
   -1     4    -2
    5     2     1
```

9.3.3 Otras matrices

MATLAB incluyen algunas matrices que son útiles para atestiguar técnicas numéricas, que sirven en algoritmos computacionales o que sólo son interesantes.

pascal	crea una matriz Pascal, con el uso del triángulo de Pascal	`pascal(4)`
		`ans =`
		`1.00 1.00 1.00 1.00`
		`1.00 2.00 3.00 4.00`
		`1.00 3.00 6.00 10.00`
		`1.00 4.00 10.00 20.00`

magic	crea una matriz mágica, en la que todas las filas, todas las columnas y todas las diagonales suman el mismo valor	`magic(3)`
		`ans =`
		`8.00 1.00 6.00`
		`3.00 5.00 7.00`
		`4.00 9.00 2.00`

rosser	La matriz Rosser se usa como una matriz de valores propios de prueba. No requiere entrada.	`rosser`
		`ans =`
		`[8 x 8]`

gallery	La galería contiene más de 50 diferentes matrices de prueba.	La sintaxis de pedido para las funciones gallery es diferente para cada función. Use help para determinar cuál es correcta para sus necesidades.

RESUMEN

Una de las operaciones matriciales más comunes es la transposición, que cambia filas a columnas y columnas a filas. En los textos de matemáticas, el transpuesto se indica con un superíndice T, como en A^T. En MATLAB, se usa el apóstrofe como el operador transponer. Por tanto,

```
A'
```

es la transpuesta de **A**.

Otra operación matricial común es el producto punto, que es la suma de las multiplicaciones de arreglo de dos vectores de igual tamaño:

$$C = \sum_{i=1}^{N} A_i * B_i$$

La función MATLAB para el producto punto es

```
dot(A,B)
```

Similar al producto punto es la multiplicación matricial. Cada elemento en el resultado de una multiplicación matricial es un producto punto:

$$C_{i,j} = \sum_{k=1}^{N} A_{i,k} B_{k,j}$$

La multiplicación matricial usa el operador asterisco en MATLAB, de modo que

```
C = A*B
```

indica que la matriz A se multiplica por la matriz B en concordancia con las reglas del álgebra matricial. La multiplicación matricial no es conmutativa; esto es,

$$AB \neq BA$$

Elevar una matriz a una potencia es similar a múltiples pasos de multiplicación:

$$A^3 = AAA$$

Dado que una matriz debe elevarse al cuadrado para multiplicarse por ella misma, sólo las matrices cuadradas se pueden elevar a una potencia. Cuando las matrices se elevan a potencias no enteras, el resultado es una matriz de números complejos.

Una matriz por su inverso es la matriz identidad:

$$AA^{-1} = I$$

MATLAB proporciona dos técnicas para determinar una matriz inversa: la función **inv**, con lo cual

```
inv_of_A=inv(A)
```

y elevar la matriz a la potencia -1, dada por

```
inv_of_A = A^-1
```

Si el determinante de una matriz es cero, la matriz es singular y no tiene inverso. La función MATLAB que se usa para encontrar el determinante es

```
det(A)
```

Además de calcular productos punto, MATLAB contiene una función que calcula el producto cruz de dos vectores en el espacio. El producto cruz con frecuencia se llama producto vectorial porque regresa un vector:

$$C = A \times B$$

El producto cruz produce un vector que está en ángulos rectos (normal) a los dos vectores de entrada, una propiedad llamada ortogonalidad. Los productos cruz se pueden considerar como el determinante de una matriz compuesta de los vectores unitarios en las direcciones x, y y z y los dos vectores de entrada:

$$C = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{vmatrix}$$

La sintaxis MATLAB para calcular un producto cruz usa la función **cross**:

```
C = cross(A,B)
```

Un uso común de la matriz inversa es resolver sistemas de ecuaciones lineales. Por ejemplo, el sistema

$$\begin{array}{rrrcr} 3x & +2y & -z & = & 10 \\ -x & +3y & +2z & = & 5 \\ x & -y & -z & = & -1 \end{array}$$

se puede expresar con matrices como

$$\mathbf{AX} = \mathbf{B}$$

Para resolver este sistema de ecuaciones con MATLAB, podría multiplicar **B** por el inverso de **A**:

```
X = inv(A)*B
```

Sin embargo, esta técnica es menos eficiente que la eliminación gaussiana, que se logra en MATLAB con el uso de la división izquierda:

```
X = A\B
```

MATLAB incluye algunas matrices especiales que se pueden usar para realizar cálculos más fáciles o para probar técnicas numéricas. Por ejemplo, las funciones **ones** y **zeros** se pueden usar para crear matrices de unos y ceros, respectivamente. Las funciones **pascal** y **magic** se usan para crear matrices Pascal y matrices mágicas, respectivamente, ninguna de las cuales tiene algún uso computacional particular, pero son matemáticamente interesantes. La función gallery contiene más de 50 matrices especialmente formuladas para probar técnicas numéricas.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona y describe brevemente todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres especiales	
'	indica una matriz transpuesta
*	multiplicación matricial
\	división izquierda matricial
^	exponenciación matricial

Comandos y funciones	
`cross`	calcula el producto cruz
`det`	calcula el determinante de una matriz
`dot`	calcula el producto punto
`eye`	genera una matriz identidad
`gallery`	contiene matrices muestra
`inv`	calcula el inverso de una matriz
`magic`	crea una matriz "mágica"
`ones`	crea una matriz que contiene todos unos
`pascal`	crea una matriz Pascal
`size`	determina el número de filas y columnas en una matriz
`zeros`	crea una matriz que contiene todos ceros

TÉRMINOS CLAVE

determinante
eliminación gaussiana
inverso
matriz identidad
multiplicación matricial
normal
ortogonal
producto cruz
producto punto
singular
sistema de ecuaciones
transponer
vector unitario

PROBLEMAS

Productos punto

9.1 Calcule el producto punto de los siguientes pares de vectores y luego demuestre que

$$A \bullet B = B \bullet A$$

(a) **A** = [1 3 5], **B** = [−3 −2 4]
(b) **A** = [0 −1 −4 −8], **B** = [4 −2 −3 24]

9.2 Calcule la masa total de los componentes que se muestra en la tabla 9.3, con un producto punto.

9.3 Use un producto punto y la lista de compras de la tabla 9.4 para determinar su cuenta total en la tienda.

9.4 Los calorímetros de bomba se usan para determinar la energía liberada durante reacciones químicas. La capacidad calorífica total de un calorímetro de bomba se define como la suma de los productos de la masa de cada componente y la capacidad calorífica específica de cada componente, o

$$\text{CP} = \sum_{i=1}^{n} m_i C_i$$

donde

m_i = masa del componente i, g
C_i = capacidad calorífica del componente, i, J/g, K
CP = capacidad calorífica total, J/K

Encuentre la capacidad calorífica total de un calorímetro de bomba con los datos térmicos que se muestran en la tabla 9.5.

Tabla 9.3 Propiedades de masa de componente

Componente	Densidad	Volumen
Propelente	1.2 g/cm^3	700 cm^3
Acero	7.8 g/cm^3	200 cm^3
Aluminio	2.7 g/cm^3	300 cm^3

Tabla 9.4 Lista de compras

Artículo	Número necesario	Costo
Leche	2 galones	$3.50 por galón
Huevos	1 docena	$1.25 por docena
Cereal	2 cajas	$4.25 por caja
Sopa	5 latas	$1.55 por lata
Galletas	1 paquete	$3.15 por paquete

Tabla 9.5 Datos térmicos

Componente	Masa	Capacidad calorífica
Acero	250 g	0.45 J/gK
Agua	100 g	4.2 J/gK
Aluminio	10 g	0.90 J/gK

9.5 Los compuestos orgánicos están constituidos principalmente de carbono, hidrógeno y oxígeno, y con frecuencia se llaman hidrocarburos por dicha razón. El peso molecular (MW) de cualquier compuesto es la suma de los productos del número de átomos de cada elemento (Z) y el peso atómico (AW) de cada elemento presente en el compuesto.

$$\text{MW} = \sum_{i=1}^{n} \text{AW}_i \cdot Z_i$$

Los pesos atómicos del carbono, hidrógeno y oxígeno son aproximadamente 12, 1 y 16, respectivamente. Use un producto punto para determinar el peso molecular del etanol (C_2H_5OH), que tiene dos carbonos, un oxígeno y seis átomos de hidrógeno.

9.6 Con frecuencia es útil pensar que el aire es una sola sustancia con un peso molecular (masa molar) determinada por un promedio ponderado de los pesos moleculares de los diferentes gases presentes en el aire. Con poco error, se puede estimar el peso molecular del aire sólo con el uso de nitrógeno, oxígeno y dióxido de carbono en los cálculos. Use un producto punto y la tabla 9.6 para aproximar el peso molecular del aire.

Multiplicación matricial

9.7 Calcule el producto matricial **A*B** de los siguientes pares de matrices:

(a) $A = \begin{bmatrix} 12 & 4 \\ 3 & -5 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 12 \\ 0 & 0 \end{bmatrix}$

(b) $A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} -2 & 4 \\ 3 & 8 \\ 12 & -2 \end{bmatrix}$

Demuestre que **A*B** no es lo mismo que **B*A**.

9.8 Usted y un amigo van a una tienda. Sus listas son las que se muestran en la tabla 9.7.

Tabla 9.6 Composición del aire

Compuesto	Fracción en aire	Peso molecular
Nitrógeno, N_2	0.78	28 g/mol
Oxígeno, O_2	0.21	32 g/mol
Dióxido de carbono, CO_2	0.01	44 g/mol

Tabla 9.7 Lista de compras de Ann y Fred

Artículo	Cantidad que necesita Ann	Cantidad que necesita Fred
Leche	2 galones	3 galones
Huevos	1 docena	2 docenas
Cereal	2 cajas	1 caja
Sopa	5 latas	4 latas
Galletas	1 paquete	3 paquetes

Los artículos tienen los siguientes costos:

Artículo	Costo
Leche	\$3.50 por galón
Huevos	\$1.25 por docena
Cereal	\$4.25 por caja
Sopa	\$1.55 por lata
Galletas	\$3.15 por paquete

Encuentre la factura total para cada comprador.

9.9 Con un calorímetro de bomba se realizó una serie de experimentos. En cada experimento se usó una cantidad diferente de agua. Calcule la capacidad calorífica total para el calorímetro en cada uno de los experimentos, mediante multiplicación matricial, los datos de la tabla 9.8 y la información acerca de la capacidad calorífica que sigue a la tabla.

Tabla 9.8 Propiedades térmicas de un calorímetro de bomba

Experimento núm.	Masa de agua	Masa de acero	Masa de aluminio
1	110 g	250 g	10 g
2	100 g	250 g	10 g
3	101 g	250 g	10 g
4	98.6 g	250 g	10 g
5	99.4 g	250 g	10 g

Componente	Capacidad calorífica
Acero	0.45 J/gK
Agua	4.2 J/gK
Aluminio	0.90 J/gK

Tabla 9.9 Composición de alcoholes

Nombre	Carbono	Hidrógeno	Oxígeno
Metanol	1	4	1
Etanol	2	6	1
Propanol	3	8	1
Butanol	4	10	1
Pentanol	5	12	1

9.10 El peso molecular (MW) de cualquier compuesto es la suma de los productos del número de átomos de cada elemento (Z) y el peso atómico (AW) de cada elemento presente en el compuesto, o

$$\text{MW} = \sum_{i=1}^{n} \text{AW}_i \cdot Z_i$$

En la tabla 9.9 se mencionan las composiciones de los primeros cinco alcoholes de cadena recta. Use los pesos atómicos del carbono, hidrógeno y oxígeno (12, 1 y 16, respectivamente) y la multiplicación matricial para determinar el peso molecular (más correctamente llamada masa molar) de cada alcohol.

Exponenciación matricial

9.11 Dado el arreglo

$$A = \begin{bmatrix} -1 & 3 \\ 4 & 2 \end{bmatrix}$$

(a) Eleve **A** a la segunda potencia mediante exponenciación de arreglo. (Consulte **help** si es necesario.)
(b) Eleve **A** a la segunda potencia mediante exponenciación matricial.
(c) Explique por qué las respuestas son diferentes.

9.12 Cree un arreglo 3×3 llamado **A** mediante la función **pascal**:

```
pascal(3)
```

(a) Eleve **A** a la tercera potencia mediante exponenciación de arreglo. (Consulte **help** si es necesario.)
(b) Eleve **A** a la tercera potencia mediante exponenciación matricial.
(c) Explique por qué las respuestas son diferentes.

Determinantes e inversos

9.13 Dado el arreglo **A** = [−1 3; 4 2], calcule el determinante de **A** tanto a mano como con MATLAB.

9.14 Recuerde que no todas las matrices tienen inverso. Una matriz es singular (es decir: no tiene inverso) si su determinante es igual a 0 (es decir, $|A| = 0$). Use la función determinante para probar si cada una de las siguientes matrices tiene inverso:

$$A = \begin{bmatrix} 2 & -1 \\ 4 & 5 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}, \quad C = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 2 \\ 5 & -4 & 0 \end{bmatrix}$$

Si existe un inverso, calcúlelo.

Productos cruz

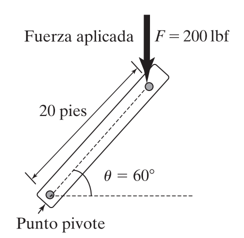
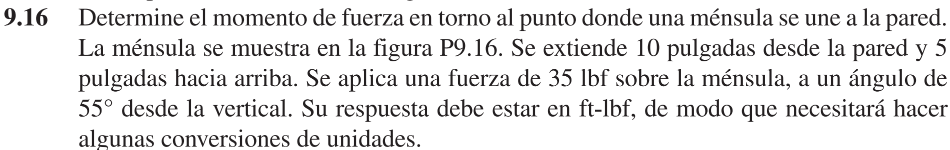


Figura P9.15
Momento de fuerza que actúa sobre una palanca en torno al origen.

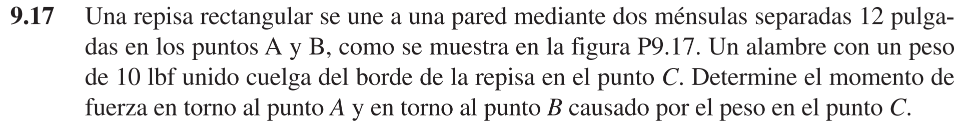
9.15 Calcule el momento de fuerza en torno al punto pivote para la palanca que se muestra en la figura P9.15. Necesitará usar trigonometría para determinar los componentes *x* y *y* tanto del vector posición como del vector fuerza. Recuerde que el momento de fuerza se puede calcular como el producto cruz

$$\mathbf{M}_0 = \mathbf{r} \times \mathbf{F}$$

Una fuerza de 200 lbf se aplica verticalmente en una posición a 20 pies sobre la palanca. La palanca se ubica en un ángulo de 60° desde la horizontal.



9.16 Determine el momento de fuerza en torno al punto donde una ménsula se une a la pared. La ménsula se muestra en la figura P9.16. Se extiende 10 pulgadas desde la pared y 5 pulgadas hacia arriba. Se aplica una fuerza de 35 lbf sobre la ménsula, a un ángulo de 55° desde la vertical. Su respuesta debe estar en ft-lbf, de modo que necesitará hacer algunas conversiones de unidades.



9.17 Una repisa rectangular se une a una pared mediante dos ménsulas separadas 12 pulgadas en los puntos A y B, como se muestra en la figura P9.17. Un alambre con un peso de 10 lbf unido cuelga del borde de la repisa en el punto *C*. Determine el momento de fuerza en torno al punto *A* y en torno al punto *B* causado por el peso en el punto *C*.

Puede formular este problema al resolverlo dos veces, una por cada ménsula, o al crear una matriz 2 × 3 para el vector posición y otra matriz 2 × 3 para el vector fuerza. Cada fila debe corresponder a una ménsula diferente. La función **cross** regresará un resultado 2 × 3, donde cada fila corresponda al momento en torno a una ménsula separada.

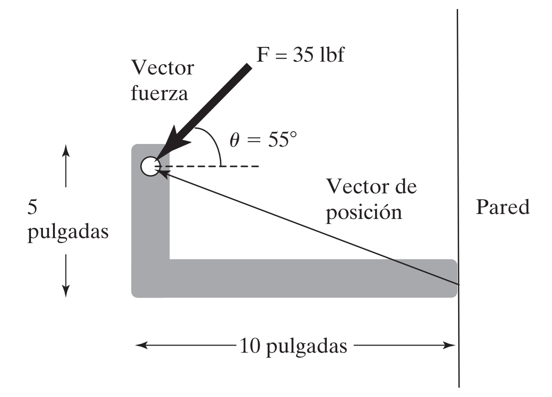


Figura P9.16
Ménsula unida a una pared.

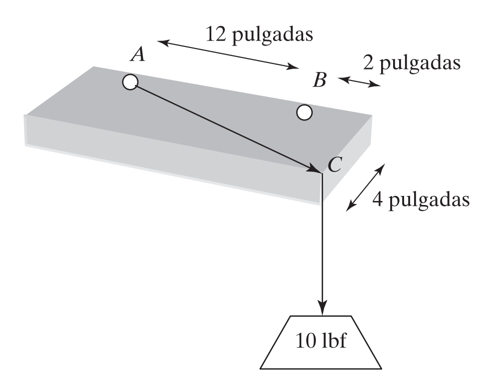


Figura P9.17
Cálculo del momento de fuerza en tres dimensiones.

Resolución de sistemas de ecuaciones lineales

9.18 Resuelva los siguientes sistemas de ecuaciones, tanto con división izquierda matricial como con el método de matriz inversa:

(a)

$$-2x + y = 3$$
$$x + y = 10$$

(b)

$$5x + 3y - z = 10$$
$$3x + 2y + z = 4$$
$$4x - y + 3z = 12$$

(c)

$$3x + y + z + w = 24$$
$$x - 3y + 7z + w = 12$$
$$2x + 2y - 3z + 4w = 17$$
$$x + y + z + w = 0$$

9.19 En general, la división izquierda matricial es más rápida y más precisa que tomar la matriz inversa. Con ambas técnicas, resuelva el siguiente sistema de ecuaciones y el tiempo de ejecución con las funciones **tic** y **toc**:

$$3x_1 + 4x_2 + 2x_3 - x_4 + x_5 + 7x_6 + x_7 = 42$$
$$2x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 + 2x_6 + 8x_7 = 32$$
$$x_1 + 2x_2 + 3x_3 + x_4 + 2x_5 + 4x_6 + 6x_7 = 12$$
$$5x_1 + 10x_2 + 4x_3 + 3x_4 + 9x_5 - 2x_6 + x_7 = -5$$
$$3x_1 + 2x_2 - 2x_3 - 4x_4 - 5x_5 - 6x_6 + 7x_7 = 10$$
$$-2x_1 + 9x_2 + x_3 + 3x_4 - 3x_5 + 5x_6 + x_7 = 18$$
$$x_1 - 2x_2 - 8x_3 + 4x_4 + 2x_5 + 4x_6 + 5x_7 = 17$$

Si tiene una computadora más nueva, puede encontrar que este problema se ejecuta tan rápidamente que no podrá detectar una diferencia entre las dos técnicas. Si es así, vea si puede formular un problema más grande para resolverlo.

9.20 En el ejemplo 9.5 se demostró que el circuito de la figura 9.5 se podría describir mediante el siguiente conjunto de ecuaciones lineales:

$$(R_2 + R_4)i_1 + (-R_2)i_2 + (-R_4)i_3 = V_1$$
$$(-R_2)i_1 + (R_1 + R_2 + R_3)i_2 + (-R_3)i_3 = 0$$
$$(-R_4)i_1 + (-R_3)i_2 + (R_3 + R_4 + R_5)i_3 = 0$$

Este conjunto de ecuaciones se resolvió con el enfoque de matriz inversa. Vuelva a hacer el problema, pero esta vez use el enfoque de división izquierda.

9.21 Considere un proceso de separación en el que una corriente de agua, etanol y metanol ingresa a una unidad de proceso. Dos corrientes salen de la unidad, cada una con cantidades variables de los tres componentes. (Véase la figura 9.21.)

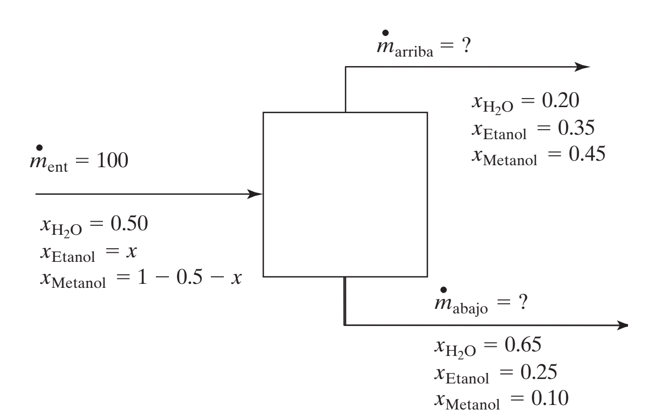


Figura P9.21
Proceso de separación con tres componentes.

Determine las tasas de flujo de masa en y afuera del sistema por arriba y abajo de la unidad de separación.

(a) Primero configure las ecuaciones de balance de materia para cada uno de los tres componentes:

<u>Agua</u>

$$(0.5)(100) = 0.2m_{\text{arriba}} + 0.65m_{\text{abajo}}$$

$$50 = 0.2m_{\text{arriba}} + 0.65m_{\text{abajo}}$$

<u>Etanol</u>

$$100x = 0.35m_{\text{arriba}} + 0.25m_{\text{abajo}}$$

$$0 = -100x + 0.35m_{\text{arriba}} + 0.25m_{\text{abajo}}$$

<u>Metanol</u>

$$100(1 - 0.5 - x) = 0.45m_{\text{arriba}} + 0.1m_{\text{abajo}}$$

$$50 = 100x + 0.45m_{\text{arriba}} + 0.1m_{\text{abajo}}$$

(b) Ordene las ecuaciones que encontró en la parte (a) en una representación matricial:

$$A = \begin{bmatrix} 0 & 0.2 & 0.65 \\ -100 & 0.35 & 0.25 \\ 100 & 0.45 & 0.1 \end{bmatrix} \qquad B = \begin{bmatrix} 50 \\ 0 \\ 50 \end{bmatrix}$$

(c) Use MATLAB para resolver el sistema lineal de tres ecuaciones.



CAPÍTULO 10

Otros tipos de arreglos

Objetivos

Después de leer este capítulo, el alumno será capaz de

- comprender los diferentes tipos de datos usados en MATLAB.
- crear y usar arreglos numéricos y carácter.
- crear arreglos multidimensionales y acceder a datos en dichos arreglos.
- crear y usar arreglos celda y estructura.

INTRODUCCIÓN

En MATLAB las matrices escalares, vectoriales y bidimensionales se usan para almacenar datos. En realidad, todas ellas son bidimensionales. Por tanto, aun cuando

```
A=1;
```

crea un escalar,

```
B=1:10;
```

crea un vector y

```
C=[1,2,3;4,5,6];
```

crea una matriz bidimensional, todas ellas son todavía arreglos bidimensionales. Note en la figura 10.1 que el tamaño de cada una de estas variables se menciona como una matriz 1×1 *bidimensional* para A, 1×10 para B y 2×3 para C. La clase que se menciona para cada una también es la misma: cada una es "double" (doble), que es abreviatura de número punto flotante de doble precisión.

MATLAB también incluye la capacidad de crear matrices multidimensionales y almacenar datos que no son dobles, como los caracteres. En este capítulo se introducirán los tipos de datos soportados por MATLAB y se explorará cómo pueden ser almacenados y usados por un programa.

10.1 TIPOS DE DATOS

El tipo de datos (también llamado clase) principal en MATLAB es el *arreglo* o *matriz*. Dentro del arreglo, MATLAB soporta algunos tipos diferentes de datos secundarios. Dado que MATLAB se escribió en C, muchos de estos tipos de datos son paralelos a los tipos de datos soportados por C. En general, todos los datos dentro de un arreglo deben ser del mismo tipo. Sin embargo, MATLAB también incluye funciones para convertir entre tipos de datos y tipos de arreglos, para almacenar diferentes tipos de datos en el mismo arreglo (arreglos celda y estructura).

Los tipos de datos que se pueden almacenar en MATLAB se mencionan en la figura 10.2. Ellos incluyen los tipos: datos numéricos, datos carácter, datos lógicos y

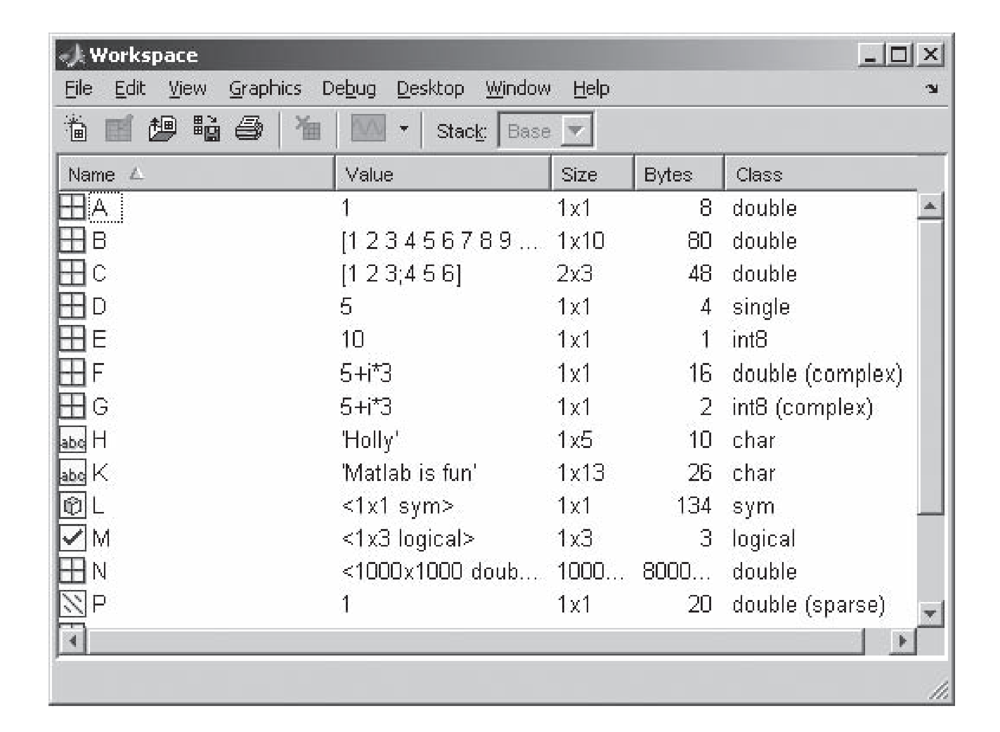


Figura 10.1
MATLAB soporta varios tipos de arreglos.

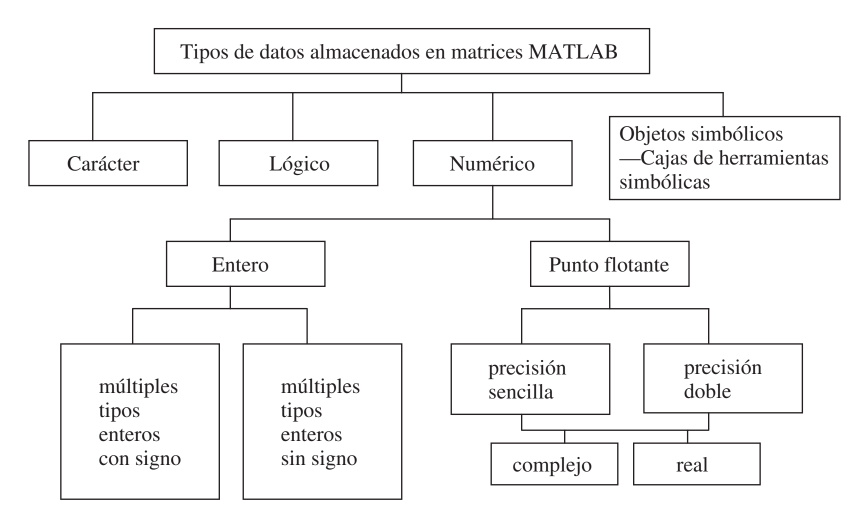


Figura 10.2
En MATLAB se pueden almacenar muchos tipos diferentes de datos.

datos simbólicos. Cada uno de estos tipos se puede almacenar en arreglos específicamente diseñados para dicho tipo de datos o en arreglos que pueden almacenar una variedad de datos. Los arreglos celda y estructura caen en esta categoría (figura 10.3).

10.1.1 Tipos de datos numéricos

Números punto flotante precisión doble

IEEE: Instituto de Ingenieros Eléctricos y Electrónicos (por sus siglas en inglés)

El tipo de datos numérico por defecto en MATLAB es el número punto flotante de precisión doble, como lo define el Estándar IEEE 754. Recuerde que, cuando se crea una variable como **A**, como en

```
A  = 1;
```

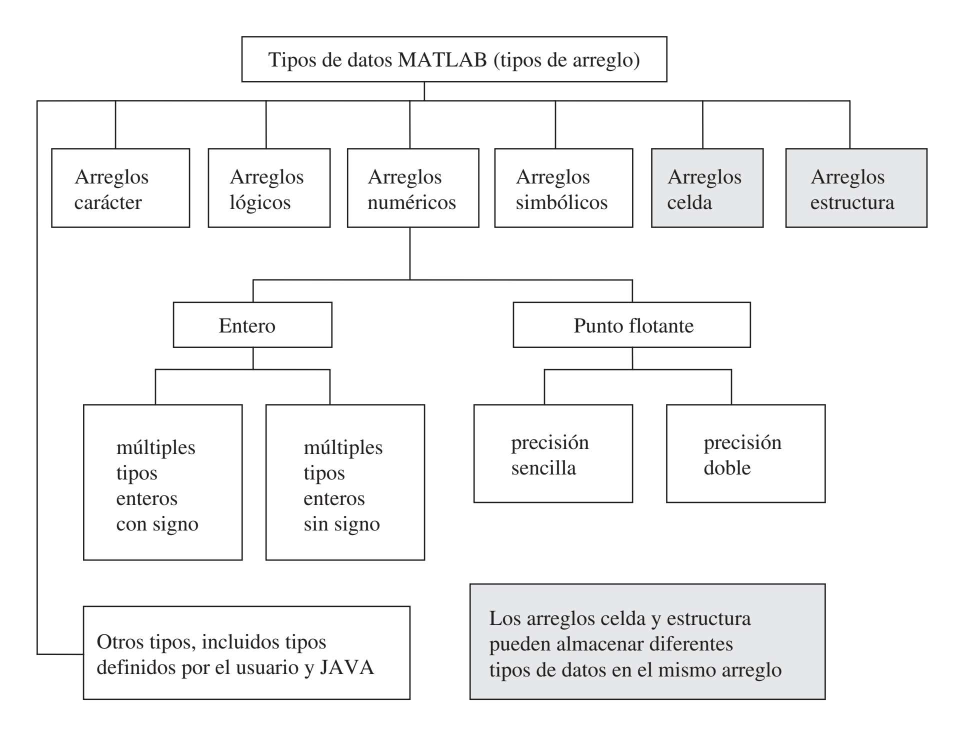


Figura 10.3
MATLAB soporta múltiples tipos de datos, todos los cuales son arreglos.

la variable mencionada en la ventana del área de trabajo y la clase es "doble", como se muestra en la figura 10.1. Note que el arreglo requiere 8 bytes de espacio de almacenamiento. Cada byte es igual a 8 bits, así que el número 1 requiere 64 bits de espacio de almacenamiento. También en la figura 10.1 note cuánto espacio de almacenamiento se requiere para las variables **B** y **C**:

Idea clave: MATLAB soporta múltiples tipos de datos.

```
B = 1:10;          C=[1,2,3; 4,5,6];
```

La **B** requiere 80 bytes, 8 para cada uno de los 10 valores almacenados, y **C** requiere 48 bytes, de nuevo 8 por cada uno de los 6 valores almacenados.

Puede usar las funciones **realmax** y **realmin** para determinar el máximo valor posible de un número punto flotante de precisión doble:

```
realmax
ans =
  1.7977e+308

realmin
ans =
  2.2251e-308
```

Si intenta ingresar un valor cuyo valor absoluto es mayor que **realmax**, o si calcula un número que está afuera de este rango, MATLAB asignará un valor de $\pm$ infinito:

```
x=5e400
x =
   Inf
```

De igual modo, si intenta ingresar un valor cuyo valor absoluto es menor que **realmin**, MATLAB asignará un valor de cero:

```
x=1e-400
x =
   0
```

Números punto flotante de precisión sencilla

Idea clave: los números de precisión sencilla requieren la mitad de espacio de almacenamiento que los números de precisión doble.

Los números punto flotante de precisión sencilla son nuevos en MATLAB 7. Sólo usan la mitad del espacio de almacenamiento de un número de precisión doble y, por tanto, sólo almacenan la mitad de información. Cada valor requiere sólo 4 bytes, o $4 \times 8 = 32$ bits de espacio de almacenamiento, como se muestra en la ventana del área de trabajo de la figura 10.1 cuando se define **D** como un número de precisión sencilla:

```
D=single(5)
D =
   5
```

Es necesario usar la función **single** para cambiar el valor 5 (que es precisión doble por defecto) a un número de precisión sencilla. De igual modo, la función **double** convertirá una variable a una doble, como en

```
double(D)
```

que cambia la variable **D** en una doble.

Dado que a los números de precisión sencilla se les asigna sólo la mitad del espacio de almacenamiento, no pueden cubrir un rango de valores tan grande como los números de precisión doble. Se pueden usar las funciones **realmax** y **realmin** para mostrar esto:

```
realmax('single')
ans =
  3.4028e+038

realmin('single')
ans =
  1.1755e-038
```

Idea clave: los números de precisión doble son apropiados para la mayoría de las aplicaciones de ingeniería.

Los ingenieros rara vez necesitarán convertir a números de precisión sencilla, porque las computadoras actuales tienen mucho espacio de almacenamiento para la mayor parte de las aplicaciones y ejecutarán la mayoría de los problemas que se planteen en tiempos extremadamente cortos. Sin embargo, en algunas aplicaciones de análisis numérico puede mejorar el tiempo de corrida de un problema largo al cambiar de precisión doble a sencilla. No obstante, note que esto tiene la desventaja de hacer del error de redondeo más que un problema.

Se puede demostrar el efecto del error de redondeo en problemas de precisión sencilla frente a precisión doble con el siguiente ejemplo: considere la serie

$$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\;\cdots\;+\frac{1}{n}+\;\cdots$$

Una serie es la suma de una secuencia de números, y esta serie particular se llama *serie armónica*, que se representa con la siguiente notación abreviada:

$$\sum_{n=1}^{\infty}\frac{1}{n}$$

La serie armónica diverge; esto es: sólo se hace más grande cuanto más términos le sume. Puede representar los primeros 10 términos de la secuencia armónica con los siguientes comandos:

```
n=1:10;
harmonic=1./n
```

Puede ver los resultados como fracciones si cambia el formato a racional:

```
format rat
harmonic =
 1   1/2   1/3   1/4   1/5   1/6   1/7   1/8   1/9   1/10
```

O puede usar el formato corto, que muestra representaciones decimales de los números:

```
format short
harmonic =
1.0000  0.5000  0.3333  0.2500  0.2000  0.1667  0.1429
        0.1250  0.1111  0.1000
```

No importa cómo se desplieguen los valores en la pantalla, se almacenan como números punto flotante de precisión doble dentro de la computadora. Al calcular las sumas parciales, se puede ver cómo cambia el valor de la suma de estos números conforme se suman más términos:

```
partial_sum=cumsum(harmonic)
partial_sum =
 Columns 1 through 6
  1.0000   1.5000   1.8333   2.0833   2.2833   2.4500
 Columns 7 through 10
  2.5929   2.7179   2.8290   2.9290
```

La función suma acumulativa (**cumsum**) calcula la suma de los valores en el arreglo hasta el número de elemento desplegado. Por tanto, en el cálculo anterior, el valor en la columna 3 es la suma parcial de los valores en las columnas de la 1 a la 3 del arreglo de entrada (en este caso, el arreglo llamado **harmonic**). No importa cuán grande se haga el arreglo armónico, las sumas parciales continúan aumentando.

El único problema con este proceso es que los valores en **harmonic** siguen haciéndose cada vez más pequeños. Eventualmente, cuando **n** es lo suficientemente grande, **1./n** es tan pequeño que la computadora no lo puede distinguir de cero. Esto ocurre mucho más rápidamente con representaciones de precisión sencilla de números que con precisión doble. Esta propiedad se puede demostrar con un gran arreglo de *n* valores:

```
n=1:1e7;
harmonic=1./n;
partial_sum=cumsum(harmonic);
```

(Probablemente calcular esto le tomará a su computadora algún tiempo.) Todos estos cálculos se realizan con números de precisión doble, porque la precisión doble es el tipo de datos por defecto en MATLAB. Ahora se podrían graficar los resultados, pero realmente hay demasiados números (10 millones, de hecho). Se puede seleccionar cada valor milésimo con el siguiente código:

```
m=1000:1000:1e7;
partial_sums_selected=partial_sum(m);
plot(partial_sums_selected)
```

Ahora se pueden repetir los cálculos, pero cambie a valores de precisión sencilla. Es posible que deba limpiar la memoria de su computadora antes de este paso, dependiendo de cuánta memoria esté disponible en su sistema. El código es

```
n=single(1:1e7);
harmonic=1./n;
```

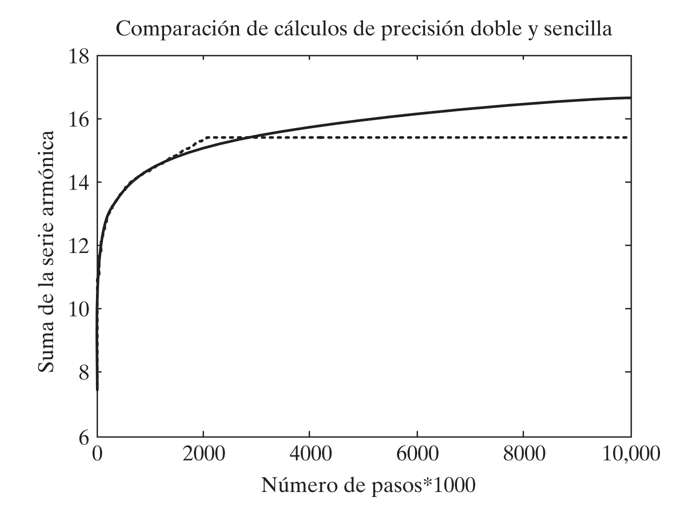


Figura 10.4
El error de redondeo degrada el cálculo de la serie armónica para números de precisión sencilla más rápido que para números de precisión doble.

```
partial_sum=cumsum(harmonic);
m=1000:1000:1e7;
partial_sums_selected=partial_sum(m);
hold on
plot(partial_sums_selected,':')
```

Idea clave: el error de redondeo es un problema más grande en los cálculos de precisión sencilla que en los de precisión doble.

Los resultados se presentan en la figura 10.4. La línea sólida representa las sumas parciales calculadas con precisión doble. La línea rayada representa las sumas parciales calculadas con precisión sencilla. El cálculo de precisión sencilla se nivela porque se alcanza el punto donde cada término sucesivo es tan pequeño que la computadora lo establece igual a cero. Para valores de precisión doble todavía no se alcanza dicho punto.

Enteros

Una novedad en MATLAB son los muchos tipos de números enteros. Tradicionalmente, los enteros se usan como números de conteo. Por ejemplo, no puede haber 2.5 personas en una habitación, y usted no puede especificar el elemento número 1.5 en un arreglo. MATLAB soporta ocho tipos diferentes de enteros, que difieren uno de otro en cuánto espacio de almacenamiento se asigna al tipo y si los valores tienen signo o no. Cuanto mayor sea el espacio de almacenamiento, más grande será el valor del número entero que puede usar. Los ocho tipos se muestran en la tabla 10.1.

Dado que 8 bits es un byte, cuando se asigna **E** como un **int8** con el código

```
E=int8(10)
E =
  10
```

requiere sólo 1 byte de almacenamiento, como se muestra en la figura 10.1.

Tabla 10.1 Tipos enteros MATLAB

entero con signo de 8 bits	**int8**	entero sin signo de 8 bits	**uint8**
entero con signo de 16 bits	**int16**	entero sin signo de 16 bits	**uint16**
entero con signo de 32 bits	**int32**	entero sin signo de 32 bits	**uint32**
entero con signo de 64 bits	**int64**	entero sin signo de 64 bits	**uint64**

Se puede determinar el valor máximo de cualquiera de los tipos enteros con el uso de la función **intmax**. Por ejemplo, el código

```
intmax('int8')
ans =
  127
```

indica que el valor máximo de un entero con signo de 8 bits es 127.

Los cuatro tipos de entero con signo asignan espacio de almacenamiento para especificar si el número es más o menos. Los cuatro tipos de entero sin signo suponen que el número es positivo y, por tanto, no necesita almacenar dicha información, lo que deja más espacio para almacenar valores numéricos. El código

Idea clave: con frecuencia, los datos enteros se usan para almacenar datos de imagen.

```
intmax('uint8')
ans =
  255
```

revela que el valor máximo de un entero sin signo de 8 bits es 255.

Los arreglos enteros encuentran uso en arreglos que se utilizan para almacenar información de imagen. Dichos arreglos usualmente son muy grandes, pero con frecuencia se usa un número limitado de colores para crear la imagen. Almacenar la información como arreglos enteros sin signo reduce dramáticamente el requerimiento de almacenamiento.

Números complejos

El tipo de almacenamiento por defecto para números complejos es doble; sin embargo, se necesita el doble de espacio de almacenamiento porque se deben almacenar los componentes real e imaginario.

```
F=5+3i;
```

Por tanto, se requieren 16 bytes (= 128 bits) para almacenar un número complejo doble. Los números complejos también se pueden almacenar como sencillos o enteros (véase la figura 10.1), como ilustra el siguiente código:

```
G =int8(5+3i);
```

Ejercicio de práctica 10.1

1. Ingrese la siguiente lista de números en arreglos de cada uno de los tipos de datos numéricos [1,4,6;3,15,24;2,3,4]:
   a. punto flotante precisión doble: llame a este arreglo **A**
   b. punto flotante precisión sencilla: llame a este arreglo **B**
   c. entero con signo (escoja un tipo): llame a este arreglo **C**
   d. entero sin signo (escoja un tipo): llame a este arreglo **D**
2. Cree una nueva matriz **E** al sumar **A** a **B**:

   E = A + B

   ¿De qué tipo de dato es el resultado?
3. Defina **x** como un tipo de datos entero igual a 1 y **y** como un tipo de datos entero igual a 3.
   a. ¿Cuál es el resultado del cálculo **x/y**?
   b. ¿Cuál es el tipo de datos del resultado?

c. ¿Qué ocurre cuando realiza la división, cuando **x** se define como el entero 2 y **y** como el entero 3?

4. Use **intmax** para determinar cuál es el número más grande que puede definir para cada uno de los tipos de datos numéricos. (Asegúrese de incluir los ocho tipos de datos enteros.)
5. Use MATLAB para determinar cuál es el número más pequeño que puede definir para cada uno de los tipos de datos numéricos. (Asegúrese de incluir los ochos tipos de datos enteros.)

10.1.2 Datos carácter y cadena

Idea clave: cada carácter, incluidos espacios, es un elemento separado en un arreglo carácter.

Además de almacenar números, MATLAB puede almacenar información carácter. Los apóstrofes se usan para identificar una cadena y diferenciarla de un nombre de variable. Cuando usted escribe la cadena

```
H='Holly';
```

se crea un arreglo carácter 1 × 5. Cada letra es un elemento separado del arreglo, como se indica por el código

```
H(5)
ans =
  y
```

Cualquier cadena representa un arreglo carácter en MATLAB. Por tanto,

```
K = 'MATLAB is fun'
```

ASCII: American Standard Code for Information (código estándar americano para información), código estándar para intercambio de información entre computadoras

EBCDIC: Extended Binary Coded Decimal Interchange Code (código ampliado de intercambio decimal codificado a binario); código estándar para intercambio de información entre computadoras

binario: esquema de codificación que usa sólo ceros y unos

se convierte en un arreglo carácter 1 × 13. Note que los espacios entre las palabras cuentan como caracteres. Note también que el nombre de columna en la figura 10.1 despliega un símbolo que contiene las letras "abc", que indican que **H** y **K** son arreglos carácter. Cada carácter en un arreglo carácter requiere 2 bytes de espacio de almacenamiento.

Toda la información en las computadoras se almacena como una serie de ceros y unos. Existen dos esquemas de codificación principales para hacer esto, llamados ASCII y EBCDIC. La mayoría de las computadoras pequeñas usan el esquema de codificación ASCII, mientras que muchas unidades centrales (mainframes) y supercomputadoras usan EBCDIC. Puede pensar en la serie de ceros y unos como un binario, o número de base 2. En este sentido, toda la información de computadora se almacena numéricamente. Todo número de base 2 tiene un equivalente decimal. En la tabla 10.2 se muestran algunos números en cada base.

Tabla 10.2 Conversiones binario a decimal

Base 2 (binario)	Base 10 (decimal)
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8

Cada carácter ASCII (o EBCDIC) almacenado tiene tanto representación binaria como equivalente decimal. Cuando se pide a MATLAB cambiar un carácter a uno doble, el número que se obtiene es el equivalente decimal en el sistema de codificación ASCII. Por tanto, se puede tener

```
double('a')
ans =
  97
```

Por el contrario, cuando se usa la función **char** en uno doble, se obtiene el carácter representado por dicho número decimal en ASCII; por ejemplo,

```
char(98)
ans =
  b
```

Si, por otra parte, se intenta crear una matriz que contenga tanto información numérica como información carácter, MATLAB convierte todos los datos a información carácter:

```
['a',3]
ans =
  a□
```

(El símbolo rectangular es el equivalente ASCII del número decimal 3.)

Por otra parte, si se intenta realizar cálculos matemáticos con información numérica y carácter, MATLAB convierte el carácter a su equivalente decimal:

```
'a' + 3
ans =
  100
```

Dado que el equivalente decimal de **'a'** es 97, el problema se convierte en

$$97 + 3 = 100$$

Ejercicio de práctica 10.2

1. Cree un arreglo carácter que conste de las letras de su nombre.
2. ¿Cuál es el equivalente decimal de la letra *g*?
3. Las letras mayúsculas y minúsculas tienen una separación de 32 en equivalente decimal. (Las mayúsculas están primero.) Con funciones anidadas, convierte la cadena 'matlab' en el equivalente mayúsculo, 'MATLAB'.

10.1.3 Datos simbólicos

La caja de herramientas simbólica usa datos simbólicos para realizar cálculos algebraicos simbólicos. Una forma de crear una variable simbólica es usar la función **sym**:

```
L=sym('x^2-2')
L =
  x^2-2
```

Los requisitos de almacenamiento para un objeto simbólico dependen de cuán grande sea el objeto. Sin embargo, note en la figura 10.1 que **L** es un arreglo 1×1. Subsecuentes objetos

simbólicos se pueden agrupar en un arreglo de expresiones matemáticas. El icono variable simbólica que se muestra en la columna izquierda de la figura 10.1 es un cubo.

10.1.4 Datos lógicos

Idea clave: los programas de cómputo usan el número 0 para indicar falso y el número 1 para indicar verdadero.

Los arreglos lógicos pueden parecer como arreglos de unos y ceros porque MATLAB (así como otros lenguajes de computación) usan dichos números para denotar verdadero y falso:

```
M=[true,false,true]
M =
  1   0   1
```

No obstante, usualmente uno no crea arreglos lógicos de esta forma. Por lo general, son resultado de operaciones lógicas. Por ejemplo,

```
x=1:5;
y=[2,0,1,9,4];
z=x>y
```

regresa

```
z =
  0   1   1   0   1
```

Esto se puede interpretar como que **x>y** es falso para los elementos 1 y 4, y verdadero para los elementos 2, 3 y 5. Estos arreglos se usan en funciones lógicas y usualmente incluso no los ve el usuario. Por ejemplo,

```
find(x>y)

ans =
  2   3   5
```

dice que los elementos 2, 3 y 5 del arreglo **x** son mayores que los correspondientes elementos del arreglo **y**. Por tanto, no tiene que analizar los resultados de la operación lógica usted mismo. El icono que representa arreglos lógicos es una marca de verificación (figura 10.1).

10.1.5 Arreglos esparcidos

Tanto los arreglos de precisión doble como los lógicos se pueden almacenar en matrices llenas o como matrices esparcidas. Las matrices esparcidas están "escasamente pobladas", lo que significa que muchos o la mayoría de los valores en el arreglo son cero. (Las matrices identidad son ejemplos de matrices esparcidas.) Si se almacenan arreglos esparcidos en el formato de matriz llena, toma 8 bytes de almacenamiento por cada valor de dato, sea cero o no. El formato de matriz esparcida sólo almacena los valores distintos de cero y recuerda dónde están, estrategia que ahorra mucho espacio.

Por ejemplo, defina una matriz identidad 1000×1000, que es una matriz de 1 millón de elementos:

```
N = eye(1000);
```

A 8 bytes por elemento, toma 8 MB almacenar esta matriz. Si se le convierte a una matriz esparcida, se puede ahorrar algo de espacio. El código para hacer esto es

```
P= sparse(A);
```

¡Note en la ventana del área de trabajo que el arreglo **P** sólo requiere 16,004 bytes! Las matrices esparcidas se pueden usar en cálculos tal como las matrices llenas. El icono que representa un arreglo esparcido es un grupo de líneas diagonales (figura 10.1).

10.2 ARREGLOS MULTIDIMENSIONALES

Cuando surge la necesidad de almacenar datos en arreglos multidimensionales (más que bidimensionales), MATLAB representa los datos con páginas adicionales. Suponga que le gustaría combinar los siguientes cuatro arreglos bidimensionales en un arreglo tridimensional:

Idea clave: MATLAB soporta arreglos en más de dos dimensiones.

```
x=[1,2,3;4,5,6];
 y=10*x;
 z=10*y;
 w=10*z;
```

Necesita definir cada página por separado:

```
my_3D_array(:,:,1)=x;
my_3D_array(:,:,2)=y;
my_3D_array(:,:,3)=z;
my_3D_array(:,:,4)=w;
```

Lea cada uno de los enunciados previos como todas la filas, todas las columnas, página 1, etcétera.

Cuando llama **my_3D_array** con el código

```
my_3D_array
```

el resultado es

```
my_3D_array
my_3D_array(:,:,1) =
  1   2   3
  4   5   6
my_3D_array(:,:,2) =
  10  20  30
  40  50  60
my_3D_array(:,:,3) =
 100  200  300
 400  500  600
my_3D_array(:,:,4) =
   1000     2000     3000
   4000     5000     6000
```

Un arreglo multidimensional se puede visualizar como se muestra en la figura 10.5. En una forma similar se pueden crear arreglos de dimensiones incluso mayores.

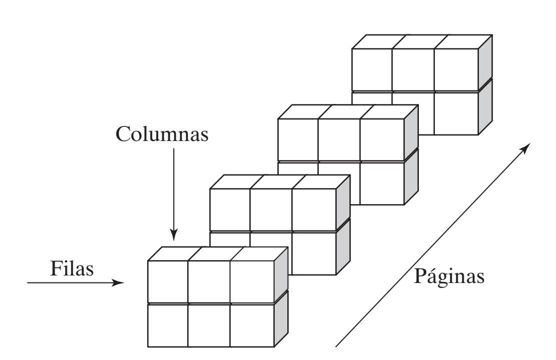


Figura 10.5
Los arreglos multidimensionales se agrupan en páginas.

Ejercicio de práctica 10.3

1. Cree un arreglo tridimensional que consista en un cuadrado mágico de 3 × 3, una matriz de ceros 3 × 3, y una matriz de unos 3 × 3.
2. Use triple indexación como **A(m,n,p)** para determinar cuál número está en la fila 3, columna 2, página 1 de una matriz creada en el problema 1.
3. Encuentre todos los valores en la fila 2, columna 3 (en todas las páginas) de la matriz.
4. Encuentre todos los valores en todas las filas y páginas de la columna 3 de la matriz.

10.3 ARREGLOS CARÁCTER

Se pueden crear arreglos carácter bidimensionales sólo si el número de elementos en cada fila es el mismo. Por tanto, una lista de nombres como la siguiente no funcionará, porque cada nombre tiene un número diferente de caracteres:

```
Q =['Holly';'Steven';'Meagan';'David';'Michael';'Heidi']

??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.
```

La función **char** "ajusta" un arreglo carácter con espacios, de modo que cada fila tenga el mismo número de elementos:

```
Q =char('Holly','Steven','Meagan','David','Michael','Heidi')

Q =
Holly
Steven
Meagan
David
Michael
Heidi
```

Q es un arreglo carácter 6 × 7. Note que, entre cada cadena en la función **char**, se usan comas.

En un arreglo carácter MATLAB no sólo se pueden almacenar caracteres alfabéticos. Cualesquiera de los símbolos o números que se encuentran en el teclado se pueden almacenar como caracteres. Se puede sacar ventaja de esta característica para crear tablas que parezca que incluyen información carácter y numérica, pero que el realidad están compuestas sólo de caracteres.

Por ejemplo, suponga que el arreglo **R** contiene puntajes de examen para los estudiantes en el arreglo carácter **Q**:

```
R=[98;84;73;88;95;100]

R =

  98
  84
  73
  88
  95
 100
```

Si se intenta combinar estos dos arreglos, se obtendrá un resultado extraño porque son dos tipos de datos diferentes:

```
table=[Q,R]
table =
Holly  b
Steven T
Meagan I
David  X
Michael_
Heidi  d
```

Los valores de doble precisión en **R** se usaron para definir caracteres sobre la base de su equivalente ASCII. Cuando en el mismo arreglo se usan doubles y chars, MATLAB convierte toda la información a chars. Esto es confuso pues, cuando se combinan caracteres y datos numéricos en cálculos matemáticos, MATLAB convierte la información carácter a información numérica.

La función **num2str** (número a cadena) le permite convertir la matriz **R** doble a una matriz compuesta de datos carácter:

```
S=num2str(R)
S =
 98
 84
 73
 88
 95
100
```

R y **S** se parecen, pero si verifica la ventana del área de trabajo (figura 10.6), verá que **R** es un arreglo double 6×1 y **S** es el arreglo char 6×3 que se muestra abajo.

espacio	9	8
espacio	8	4
espacio	7	3
espacio	8	8
espacio	9	5
1	0	0

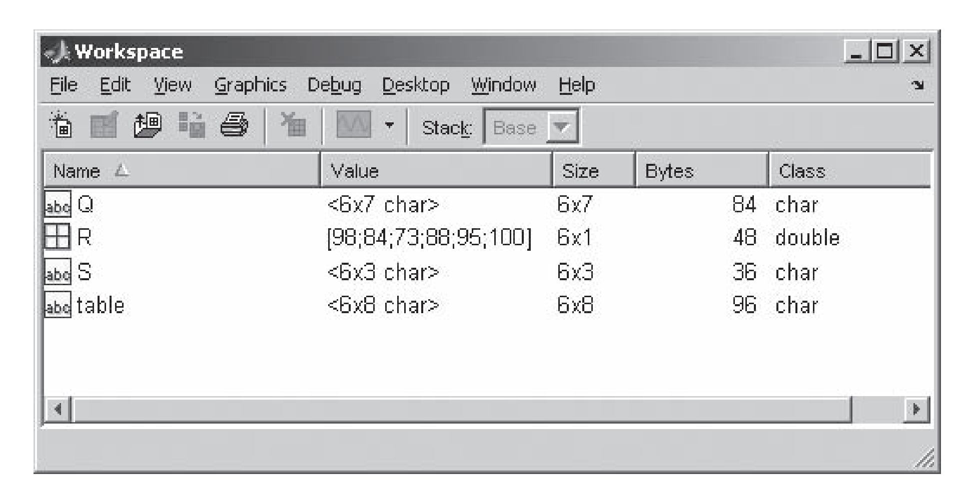


Figura 10.6
Los datos carácter y numérico se pueden combinar en un solo arreglo al cambiar los valores numéricos a caracteres con la función **num2str**.

Ahora se puede combinar **Q**, el arreglo carácter de nombres, con **S**, el arreglo carácter de puntajes:

```
table=[Q,S]

table =
Holly     98
Steven    84
Meagan    73
David     88
Michael   95
Heidi    100
```

Se muestran los resultados en la fuente monospace, que está uniformemente espaciada. Usted puede controlar la fuente que usa MATLAB; si elige una fuente proporcional, como Times New Roman, sus columnas no se alinearán.

También se podría usar la función **disp** para desplegar los resultados:

```
disp([L,N])

Holly     98
Steven    84
Meagan    73
David     88
Michael   95
Heidi    100
```

Sugerencia

Ponga un espacio después de su cadena más larga, de modo que, cuando cree un arreglo carácter acolchado, habrá un espacio entre la información carácter y la información numérica que convirtió a datos carácter.

Idea clave: combine arreglos carácter y numérico con la función **num2str** para crear nombres de archivo de datos.

Una aplicación útil de los arreglos carácter y la función **num2str** es la creación de nombres de archivo. Hay ocasiones en que usted quiere guardar datos en archivos **.dat** o **.mat**, pero no sabe por anticipado cuántos archivos requerirá. Una solución sería nombrar sus archivos con el siguiente patrón:

```
my_data1.dat
my_data2.dat
my_data3.dat etc.
```

Imagine que usted carga en MATLAB un archivo de tamaño desconocido, llamado **some_data**, y le gustaría crear nuevos archivos, cada uno compuesto de una sola columna de **some_data**:

```
load some_data
```

Puede determinar cuán grande es el archivo con el uso de la función **size**:

```
[rows,cols]=size(some_data)
```

Si quiere almacenar cada columna de los datos en su propio archivo, necesitará un nombre de archivo para cada columna. Puede hacer esto en un bucle **for**, usando la forma de función del comando **save**:

```
for k=1:cols
   file_name=['my_data',num2str(k)]
```

```
    data=some_data(:,k) '
    save(file_name,'data')
end
```

El bucle se ejecutará una vez por cada columna. Usted construye el nombre de archivo al crear un arreglo que combina caracteres y números con el enunciado

```
file_name=['my_data',num2str(k)];
```

Este enunciado establece la variable **file_name** igual a un arreglo carácter, como **my_data1** o **my_data2**, dependiendo del paso actual a través del bucle. La función **save** acepta entrada carácter. En la línea

```
save(file_name,'data')
```

file_name es una variable carácter, y **'data'** se reconoce como información carácter porque está dentro de apóstrofes. Si corre el anterior bucle **for** en un archivo que contenga una matriz 5×3 de números aleatorios, obtiene el siguiente resultado:

```
rows =
   5
cols =
   3
file_name =
my_data1
data =
   -0.4326  -1.6656   0.1253  0.2877  -1.1465
file_name =
my_data2
data =
    1.1909   1.1892  -0.0376  0.3273   0.1746
file_name =
my_data3
data =
   -0.1867   0.7258  -0.5883  2.1832  -0.1364
```

El directorio actual ahora contiene tres nuevos archivos.

Ejercicio de práctica 10.4

1. Cree una matriz carácter llamada **names** de los nombres de todos los planetas. Su matriz debe tener nueve filas.
2. Algunos de los planetas se puede clasificar como enanos rocosos y otros como gigantes gaseosos. Cree una matriz carácter llamada **type**, con la designación apropiada en cada línea.
3. Cree una matriz carácter de nueve espacios, un espacio por fila.
4. Combine sus matrices para formar una tabla que mencione los nombres de los planetas y sus designaciones, separados por un espacio.
5. Use Internet para encontrar la masa de cada uno de los planetas y almacene la información en una matriz llamada **mass**. (O use los datos del ejemplo 10.2 de la página 362.) Use la función **num2str** para convertir el arreglo numérico en un arreglo carácter y agréguelo a su tabla.

EJEMPLO 10.1

Creación de un esquema de codificación secreto simple

Mantener privada la información en una era electrónica se está volviendo cada vez más difícil. Una alternativa es codificar la información, de modo que incluso si una persona no autorizada ve la información no podrá entenderla. Las modernas técnicas de codificación son extremadamente complicadas, pero usted puede crear un código simple al sacar ventaja de la forma en que la información carácter se almacena en MATLAB. Si se agrega un valor entero constante a la información carácter, se puede transformar la cadena en algo que sea difícil de interpretar.

1. Establezca el problema.
   Codificar y decodificar una cadena de información carácter.
2. Describa las entradas y salidas.

   Entrada Información carácter ingresada desde la ventana de comandos

   Salida Información codificada

3. Desarrolle un ejemplo a mano.
   La letra minúscula *a* es equivalente al número decimal 97. Si se agrega 5 a *a* y se convierte de nuevo en carácter, se vuelve la letra *f*.
4. Desarrolle una solución MATLAB.

```
%Ejemplo 10.1
%Conmine al usuario a ingresar una cadena de información
carácter.
A=input('Ingrese una cadena de información a codificar: ')
encoded=char(A+5);
disp('Su entrada se transformó');
disp(encoded);
disp('¿Le gustaría decodificar este mensaje?');
response=menu('¿sí o no?','SÍ','NO');
switch response
  case 1
    disp(char(encoded-5));
  case 2
    disp('OK - Adiós');
end
```

5. Ponga a prueba la solución.
   Corra el programa y observe lo que ocurre. El programa le pide la entrada, que debe ingresar como cadena (dentro de apóstrofes):

```
Ingrese una cadena de información a codificar:
'I love rock and roll'
```

Una vez que oprima la tecla return, el programa responde

```
Su entrada se transformó
N%qt{j%wthp%fsi%wtqq
¿Le gustaría decodificar este mensaje?
```

Puesto que se eligió usar una opción menú para la respuesta, aparece la ventana de menú. Cuando se elige SÍ, el programa responde con




```
I love rock and roll
```

Si se elige NO, responde con

```
OK - Adiós
```

10.4 ARREGLOS CELDA

A diferencia de los arreglos numérico, carácter y simbólico, el arreglo celda puede almacenar diferentes tipos de datos dentro del mismo arreglo. Cada elemento en el arreglo también es un arreglo. Por ejemplo, considere estos tres diferentes arreglos:

Idea clave: los arreglos celda pueden almacenar información usando varios tipos de datos.

```
A=1:3;
B=['abcdefg'];
C=single([1,2,3;4,5,6]);
```

Se crearon tres arreglos separados, todos de diferente tipo de datos y tamaño. **A** es un double, **B** es un char y **C** es un single. Se les puede combinar en un arreglo celda al usar llaves como el constructor de arreglo celda (los corchetes son los constructores estándar de arreglo):

```
my_cellarray={A,B,C}
```

regresa

```
my_cellarray =
  [1x3 double]   'abcdefg'   [2x3 single]
```

Para ahorrar espacio, los arreglos grandes se mencionan sólo con información de tamaño. Puede mostrar el arreglo completo con la función **celldisp**:

```
celldisp(my_cellarray)
my_cellarray{1} =
  1   2   3
my_cellarray{2} =
abcdefg
my_cellarray{3} =
  1   2   3
```

El sistema de indexación que se usa para arreglos celda es el mismo que se usa en otros arreglos. Puede usar un índice sencillo o un esquema de indexación fila y columna. Existen dos enfoques para recuperar información de los arreglos celda: puede usar paréntesis, como en

```
my_cellarray(1)
ans =
  [1x3 double]
```

o puede usar llaves, como en

```
my_cellarray{1}
ans =
   1   2   3
```

Para acceder a un elemento particular dentro de un arreglo almacenado en un arreglo celda, debe usar una combinación de llaves y paréntesis:

```
my_cellarray{3}(1,2)
ans =
   2
```

Los arreglos celda pueden ser útiles para proyectos complicados de programación o para aplicaciones de bases de datos. Un uso común en aplicaciones de ingeniería sería almacenar todos los diversos tipos de datos de un proyecto en un nombre de variable que se pueda desensamblar y usar más tarde.

10.5 ARREGLOS ESTRUCTURA

Idea clave: los arreglos estructura pueden almacenar información usando varios tipos de datos.

Los arreglos estructura son similares a los arreglos celda. Los arreglos múltiples de diferentes tipos de datos se pueden almacenar en arreglos estructura, tal como se puede hacer en arreglos celda. No obstante, en lugar de usar indexación de contenido, a cada una de las matrices almacenadas en un arreglo estructura se le asigna una ubicación llamada *campo* (field). Por ejemplo, al usar los tres arreglos de la sección anterior en arreglos celda, a saber,

```
A=1:3;
B=['abcdefg'];
C=single([1,2,3;4,5,6]);
```

se puede crear un arreglo estructura simple llamado **my_structure**:

```
my_structure.some_numbers=A
```

que regresa

```
my_structure =
  some_numbers: [1 2 3]
```

El nombre del arreglo estructura es **my_structure**. Tiene un campo, llamado **some_numbers**. Ahora se puede agregar el contenido de la matriz carácter **B** a un segundo campo llamado **some_letters**:

```
my_structure.some_letters=B
my_structure =
  some_numbers: [1 2 3]
  some_letters: 'abcdefg'
```

Finalmente, se agregan los números de precisión sencilla de la matriz **C** a un tercer campo llamado **some_more_numbers**:

```
my_structure.some_more_numbers=C
my_structure =
  some_numbers: [1 2 3]
  some_letters: 'abcdefg'
  some_more_numbers: [2x3 single]
```

Note en la ventana del área de trabajo (figura 10.7) que la matriz estructura (llamada **struct**) es un arreglo 1×1 que contiene toda la información de las tres matrices disímiles. La estructura tiene tres campos, cada uno de los cuales contiene un tipo de datos diferente:

some_numbers	datos numéricos precisión doble
some_letters	datos carácter
some_more_numbers	datos numéricos precisión sencilla

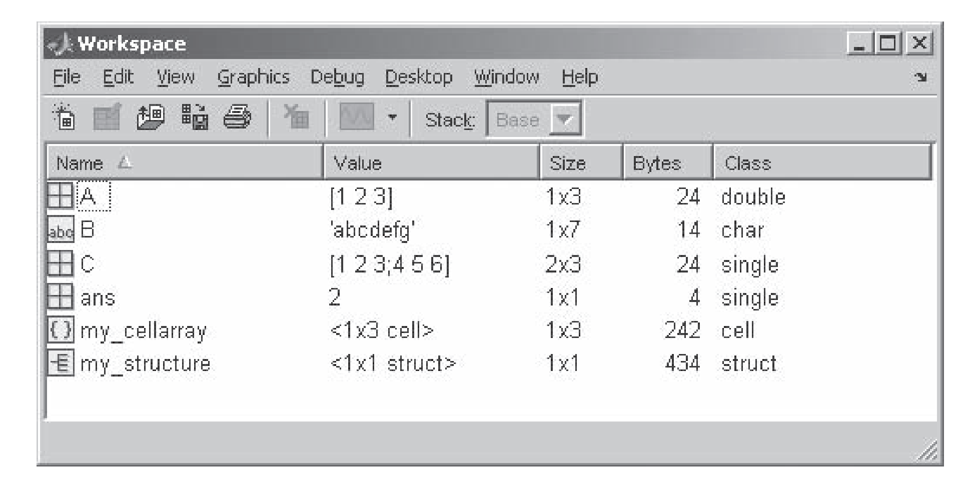


Figura 10.7
Los arreglos estructura pueden contener muchos diferentes tipos de datos.

Se puede agregar más contenido a la estructura, y expandir su tamaño, al agregar más matrices a los campos definidos:

```
my_structure(2).some_numbers=[2 4 6 8]
my_structure =
1x2 struct array with fields:
  some_numbers
  some_letters
  some_more_numbers
```

Puede acceder a la información en los arreglos estructura con el uso del nombre de matriz, nombre de campo y números índice. La sintaxis es similar a la que se usó para otros tipos de matrices. Un ejemplo es

```
my_structure(2)
ans =
        some_numbers: [2 4 6 8]
        some_letters: []
   some_more_numbers: []
```

Note que **some_letters** y **some_more_numbers** son matrices vacías, porque no se agregó información a dichos campos.

Para acceder sólo a un campo, agregue el nombre del campo:

```
my_structure(2).some_numbers
ans =
   2     4     6     8
```

Finalmente, si quiere conocer el contenido de un elemento particular en un campo, debe especificar el número índice del elemento después del nombre de campo:

```
my_structure(2).some_numbers(2)
ans =
   4
```

La función **disp** despliega los contenidos de los arreglos estructura. Por ejemplo,

```
disp(my_structure(2).some_numbers(2))
```

regresa

```
   4
```

También puede usar el editor de arreglos para acceder al contenido de un arreglo estructura (y cualquier otro arreglo, si de eso se trata). Cuando hace doble clic en el arreglo estructura en la ventana del área de trabajo, se abre el editor de arreglos (figura 10.8). Si hace doble clic en uno de los elementos de la estructura en el editor de arreglo, el editor se expande para mostrarle los contenidos de dicho elemento (figura 10.9).

Los arreglos estructura son de uso limitado en ***cálculos*** de ingeniería, pero son extremadamente útiles en aplicaciones como ***gestión de bases de datos***. Dado que usualmente grandes cantidades de datos de ingeniería se almacenan en una base de datos, el arreglo estructura es extremadamente útil para análisis de datos. Los ejemplos que siguen le ayudarán a tener una mejor idea de cómo manipular y usar arreglos estructura.

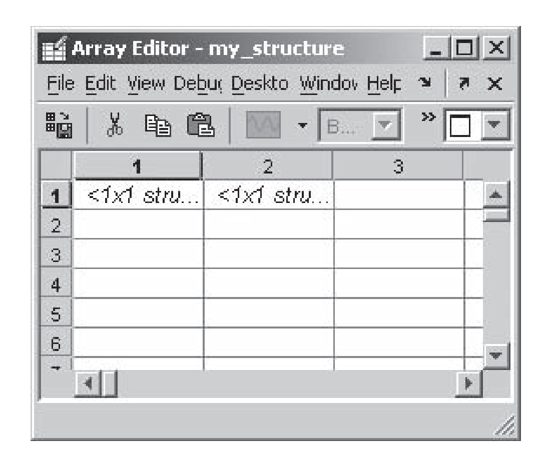


Figura 10.8
El editor de arreglo reporta el tamaño de un arreglo con la finalidad de ahorrar espacio.

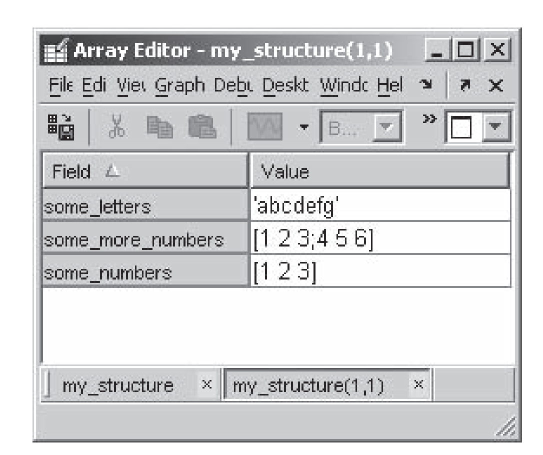


Figura 10.9
Hacer doble clic a un componente en el editor de arreglos le permite ver los datos almacenados en el arreglo.

EJEMPLO 10.2

Almacenamiento de datos planetarios con arreglos estructura

Los arreglos estructura se pueden usar en forma muy parecida a una base de datos. Puede almacenar información numérica, así como datos carácter o cualquiera de los otros tipos de datos soportados por MATLAB. Cree un arreglo estructura para almacenar información acerca de los planetas. Conmine al usuario a ingresar los datos.

1. Establezca el problema.
   Crear un arreglo estructura para almacenar datos planetarios e ingresar la información de la tabla 10.3.
2. Describa las entradas y salidas.

   Entrada

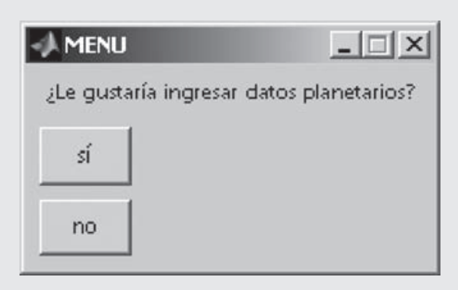
Tabla 10.3

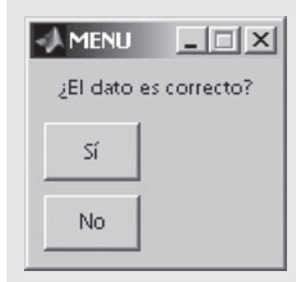
Nombre del planeta	Masa en múltiplos terrestres	Duración del año, en años terrestres	Velocidad orbital media, km/s
Mercurio	0.055	0.24	47.89
Venus	0.815	0.62	35.03
Tierra	1	1	29.79
Marte	0.107	1.88	24.13
Júpiter	318	11.86	13.06
Saturno	95	29.46	9.64
Urano	15	84.01	6.81
Neptuno	17	164.8	5.43
Plutón	0.002	247.7	4.74

   Salida

   Un arreglo estructura que almacene los datos
3. Desarrolle un ejemplo a mano.
   Desarrollar un ejemplo a mano para este problema sería difícil. En vez de ello, sería útil un diagrama de flujo.
4. Desarrolle una solución MATLAB.

```
%Ejemplo 10.2
clear,clc
%Cree una estructura con 4 campos
k=1;
planetary     %saca la información almacenada en planetary
```





He aquí una muestra de interacción en la ventana de comandos cuando corre el programa y comienza a ingresar datos:

```
Recuerde ingresar cadenas en apóstrofes
Ingrese el nombre de un planeta en apóstrofes: 'Mercurio'
Ingrese la masa planetaria en múltiplos de la masa de la Tierra:
  0.055
Ingrese la duración del año planetario en años terrestres: 0.24
Ingrese la velocidad orbital media en km/s: 47.89
ans =
     name: 'Mercurio'
     mass: 0.0550
     year: 0.2400
     velocity: 47.8900
```

5. Ponga a prueba la solución.
   Ingrese los datos y compare su arreglo con la tabla de entrada. Como parte del programa, se reportaron los valores de entrada de vuelta a la pantalla, de modo que el usuario podría verificar su precisión. Si el usuario responde que los datos no son correctos, la información se sobrescribe la siguiente vez en el bucle. También se usaron menús en lugar de respuestas libres a algunas preguntas, de modo que no habría ambigüedad en cuanto a las respuestas. Note que el arreglo estructura que se construye, llamado **planetary**, se menciona en la ventana del área de trabajo. Si hace doble clic en **planetary**, se abre el editor de arreglo y le permite ver cualquiera de los datos en el arreglo (figura 10.10). También puede actualizar cualquiera de los valores en el editor de arreglo.
   Este arreglo estructura se usará en el ejemplo 10.3 para realizar algunos cálculos. Necesitará guardar sus resultados como

```
save   planetary_information   planetary
```

   Esta secuencia de comandos guarda el arreglo estructura **planetary** en el archivo **planetary_information.mat**.

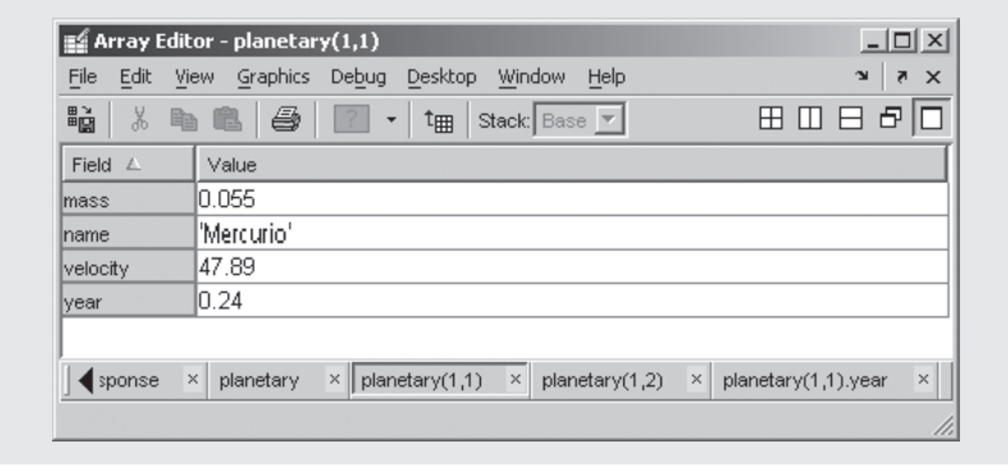


Figura 10.10
El editor de arreglos le permite ver (y cambiar) datos en el arreglo estructura.

EJEMPLO 10.3

Extracción y uso de datos desde arreglos estructura

Los arreglos estructura tienen algunas ventajas para almacenar información. Primero, usan nombres de campo para identificar componentes del arreglo. Segundo, se puede agregar información al arreglo fácilmente y siempre se asocia con un grupo. Finalmente, en los arreglos estructura es difícil mezclar accidentalmente la información. Para demostrar estas ventajas, use los datos que almacenó en el archivo **planetary_information** para completar las siguientes tareas:

- Identificar los nombres de campo en el arreglo y citarlos.
- Crear una lista de los nombres de los planetas.
- Crear una tabla que represente los datos en el arreglo estructura. Incluir los nombres de campo como encabezados de columna en la tabla.
- Calcular y reportar el promedio de los valores de velocidad orbital media.
- Encontrar el planeta más grande y reportar su tamaño y nombres.
- Encontrar y reportar el periodo orbital de Júpiter.

1. Establezca el problema.
   Crear un programa para realizar las tareas mencionadas.
2. Describa las entradas y salidas.

   Entrada **planetary_information.mat**, almacenada en el directorio actual

   Salida Crear un reporte en la ventana de comandos

3. Desarrolle un ejemplo a mano.
   Puede completar la mayoría de las tareas designadas al acceder a la información en el arreglo estructural planetary a través del editor de arreglos (véase la figura 10.11).
4. Desarrolle una solución MATLAB.

```
%Ejemplo 10.3
clear,clc
```

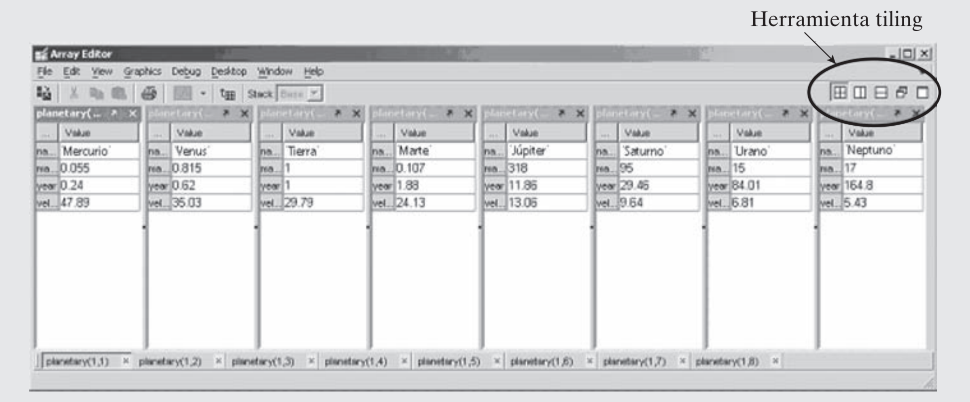


Figura 10.11
La opción tiling (mosaico) le permite ver múltiples componentes del arreglo estructura.

```
load planetary_information
%Identifique los nombres de campo en el arreglo estructura
planetary       %recuerda los contenidos del arreglo
                %estructura llamado
pause(2)
%Cree una lista de planetas en el archivo
disp('Estos nombres están bien, pero no están en el
arreglo'); planetary.name
pause(4)
fprintf('\n')   %Cree una línea vacía en la salida
%Con corchetes ponga los resultados en un arreglo
disp('Este arreglo no es muy grande');
disp('Todo corre en conjunto');
names=[planetary.name]
pause(4)
fprintf('\n')   %Cree una línea vacía en la salida
%Con char crea una lista acolchada, que es más útil
disp('Al usar un arreglo carácter acolchado se obtiene lo
  que se quiere');
names=[char(planetary.name)]
pause(4)
%Cree una tabla al crear primero arreglos carácter de todos
%los datos
disp('Estos arreglos también son arreglos carácter');
mass=num2str([planetary.mass]')
fprintf('\n')   %Cree una línea vacía en la salida
pause(4)
year=num2str([planetary.year]')
fprintf('\n')   %Cree una línea vacía en la salida
pause(2)
velocity=num2str([planetary(:).velocity]')
fprintf('\n')   %Cree una línea vacía en la salida
pause(4)
fprintf('\n')   %Cree una línea vacía en la salida
%Cree un arreglo de espacios para separar los datos
spaces=['         ']';
%Use disp para desplegar los nombres de campo
disp('El resultado global es un gran arreglo carácter');
fprintf('\n')   %Cree una línea vacía en la salida
disp('Planet  mass   year velocity');
table = [names,spaces,mass,spaces,year,spaces,velocity];
disp(table);
fprintf('\n')   %Cree una línea vacía en la salida
pause(2)
%Encuentre la velocidad orbital media planetaria promedio
MOV=mean([planetary.velocity]);
fprintf('La velocidad orbital media es %8.2 f km/s\n',MOV)
pause(1)
%Encuentre el planeta con la máxima masa
max_mass=max([planetary.mass]);
fprintf('La máxima masa es %8.2f veces la de la Tierra
  \n',max_mass)
pause(1)
%Júpiter es el planeta #5
%Encuentre el periodo orbital de Júpiter
```

```
planet_name=planetary(5).name;
planet_year=planetary(5).year;
fprintf(' %s tiene un año %6.2f veces el de la Tierra
  \n',planet_name,planet_year)
```

La mayor parte de este programa contiene comandos de formateo. Antes de intentar analizar el código, corra el programa en MATLAB y observe los resultados.

5. Ponga a prueba la solución.
   Compare la información extraída del arreglo con la información disponible del editor de arreglo. Usar el editor de arreglo se volvería difícil de manejar conforme aumenten los datos almacenados en **planetary**. Es fácil agregar nuevos campos y nueva información conforme se vuelve disponible. Por ejemplo, se podría agregar el número de lunas a la estructura existente:

```
planetary(1).moons = 0;
planetary(2).moons = 0;
planetary(3).moons = 1;
planetary(4).moons = 2;
planetary(5).moons = 60;
planetary(6).moons = 31;
planetary(7).moons = 27;
planetary(8).moons = 13;
planetary(9).moons = 1;
```

Este código agrega un nuevo campo llamado **moons** a la estructura. Se puede reportar el número de lunas para cada planeta en la ventana de comandos con el comando

```
disp([planetary.moons]);
```

RESUMEN

La principal estructura de datos de MATLAB es el arreglo. Dentro del arreglo, MATLAB permite al usuario almacenar algunos tipos diferentes de datos. El tipo de datos numérico por defecto es el número punto flotante de precisión doble, usualmente referido como double. MATLAB también soporta números punto flotante de precisión sencilla, así como ocho tipos diferentes de enteros. La información carácter también se almacena en arreglos. Los caracteres se pueden agrupar en una cadena, aunque la cadena representa un arreglo unidimensional en el que cada carácter se almacena en su propio elemento. La función **char** permite al usuario crear arreglos carácter bidimensionales a partir de cadenas de diferentes tamaños al “ajustar” el arreglo con un número apropiado de espacios en blanco. Además de datos numéricos y carácter, MATLAB incluye un tipo de datos simbólico.

Todos estos diferentes tipos de datos se pueden almacenar como arreglos bidimensionales. Los datos escalares y vectoriales en realidad se almacenan como arreglos bidimensionales: sólo tienen una fila o columna. MATLAB también permite al usuario almacenar datos en arreglos multidimensionales. Cada rebanada bidimensional de un arreglo tridimensional o mayor se llama página.

En general, los datos almacenados en un arreglo MATLAB deben ser todos del mismo tipo. Si se mezclan datos numéricos y carácter en un arreglo, los datos numéricos se cambian a datos carácter sobre la base de sus valores decimales equivalentes ASCII. Cuando se intentan cálculos en datos carácter y numéricos combinados, los datos carácter se convierten a sus equivalentes ASCII.

MATLAB ofrece dos tipos de arreglo que pueden almacenar múltiples tipos de datos al mismo tiempo: el arreglo celda y el arreglo estructura. Los arreglos celda usan llaves, { y }, como constructores de arreglo. Los arreglos estructura dependen de campos nombrados. Tanto los arreglos celda como los estructura son particularmente útiles en aplicaciones de bases de datos.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona y describe brevemente todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres especiales	
{ }	constructor arreglo celda
''	cadena de datos (información carácter)
abc	arreglo carácter
	arreglo numérico
	arreglo simbólico
	arreglo lógico
	arreglo esparcido
{ }	arreglo celda
	arreglo estructura

Comandos y funciones	
`celldisp`	despliega los contenidos de un arreglo celda
`char`	crea un arreglo carácter acolchado
`cumsum`	encuentra la suma acumulativa de los miembros de un arreglo
`double`	cambia un arreglo a un arreglo de precisión doble
`eye`	crea una matriz identidad
`format rat`	convierte el formato de despliegue a números racionales (fracciones)
`int16`	entero con signo de 16 bits
`int32`	entero con signo de 32 bits
`int64`	entero con signo de 64 bits
`int8`	entero con signo de 8 bits
`num2str`	convierte un arreglo numérico a un arreglo carácter
`realmax`	determina el número real más grande que se puede expresar en MATLAB
`realmin`	determina el número real más pequeño que se puede expresar en MATLAB
`single`	cambia un arreglo a un arreglo de precisión sencilla
`sparse`	convierte una matriz de formato lleno a una matriz de formato esparcido
`str2num`	convierte un arreglo carácter a un arreglo numérico
`uint16`	entero sin signo de 16 bits
`uint32`	entero sin signo de 32 bits
`uint64`	entero sin signo de 64 bits
`uint8`	entero sin signo de 8 bits

TÉRMINOS CLAVE

ASCII
base 2
cadena
carácter
celda
clase
datos lógicos
datos simbólicos
EBCDIC
entero
estructura
gavetas
números complejos
números punto flotante
números racionales
páginas
precisión doble
precisión sencilla
tipo de datos

PROBLEMAS

Tipos de datos numéricos

10.1 Calcule la suma (no las sumas parciales) de los primeros 10 millones de términos en la serie armónica

$$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\cdots+\frac{1}{n}+\cdots$$

con números de precisión doble y precisión sencilla. Compare los resultados. Explique por qué son diferentes.

10.2 Defina un arreglo de los primeros 10 enteros, use la designación de tipo **int8**. Use estos enteros para calcular los primeros 10 términos en la serie armónica. Explique sus resultados.

10.3 Explique por qué es mejor permitir a MATLAB tener por defecto representaciones de número punto flotante de precisión doble para la mayoría de los cálculos de ingeniería que para especificar tipos sencillos y enteros.

10.4 Los números complejos se crean automáticamente en MATLAB como resultado de los cálculos. También se pueden ingresar directamente, como la suma de un número real y uno imaginario, y se pueden almacenar como cualquiera de los tipos de datos numéricos. Defina dos variables, un número complejo de precisión sencilla y otra doble, como

```
doublea = 5 + 3i
singlea = single(5 + 3i)
```

Eleve cada uno de estos números a la centésima potencia. Explique la diferencia en sus respuestas.

Datos carácter

10.5 Use un buscador de Internet para encontrar una lista que muestre los equivalentes binarios de caracteres tanto en ASCII como en EBCDIC. Resalte brevemente las diferencias en los dos esquemas de codificación.

10.6 A veces es confuso darse cuenta de que los números se pueden representar tanto como datos numéricos como datos carácter. Use MATLAB para expresar el número 85 como un arreglo carácter.

(a) ¿Cuántos elementos hay en este arreglo?
(b) ¿Cuál es el equivalente numérico del carácter 8?
(c) ¿Cuál es el equivalente numérico del carácter 5?

Arreglos multidimensionales

10.7 Cree cada uno de los siguientes arreglos:

$$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},\quad B=\begin{bmatrix}10 & 20\\ 30 & 40\end{bmatrix},\quad C=\begin{bmatrix}3 & 6\\ 9 & 12\end{bmatrix}$$

(a) Combínelos en un gran arreglo multidimensional $2 \times 2 \times 3$ llamado **ABC**.
(b) Extraiga cada columna 1 en un arreglo 2×3 llamado **Column_A1B1C1**.
(c) Extraiga cada fila 2 en un arreglo 3×2 llamado **Row_A2B2C2**.
(d) Extraiga el valor en fila 1, columna 2, página 3.

10.8 Imagine que a una profesora universitaria le gustaría comparar cómo se desempeñan los estudiantes en una prueba que imparte cada año. Cada año, califica los datos en un arreglo bidimensional. Los datos del primero y segundo años son los siguientes:

Año 1	Pregunta 1	Pregunta 2	Pregunta 3	Pregunta 4
Estudiante 1	3	6	4	10
Estudiante 2	5	8	6	10
Estudiante 3	4	9	5	10
Estudiante 4	6	4	7	9
Estudiante 5	3	5	8	10

Año 2	Pregunta 1	Pregunta 2	Pregunta 3	Pregunta 4
Estudiante 1	2	7	3	10
Estudiante 2	3	7	5	10
Estudiante 3	4	5	5	10
Estudiante 4	3	3	8	10
Estudiante 5	3	5	2	10

(a) Cree un arreglo bidimensional llamado **year1** para los datos del primer año y otro arreglo bidimensional llamado **year2** para los datos del segundo año.
(b) Combine los dos arreglos en un arreglo tridimensional con dos páginas, llamado **testdata**.
(c) Use su arreglo tridimensional para realizar los cálculos siguientes:

- Calcule la calificación promedio para cada pregunta, para cada año y almacene los resultados en un arreglo bidimensional. (Su respuesta debe ser un arreglo 2×4 o un arreglo 4×2.)
- Calcule la calificación promedio para cada pregunta, con ***todos*** los datos.
- Extraiga los datos para la Pregunta 3 para cada año y cree un arreglo con el siguiente formato:

	Pregunta 3, Año 1	Pregunta 3, Año 2
Estudiante 1		
Estudiante 2		
etc.		

10.9 Si la profesora descrita en la pregunta anterior quiere incluir los resultados de un segundo y un tercer exámenes en el arreglo, tendría que crear un arreglo tetradimensional. (La cuarta dimensión a veces se llama *gaveta*, drawer.) Todos los datos se incluyen en un archivo llamado **test_results.mat** que consta de seis arreglos bidimensionales similares a los descritos en el problema 10.8. Los nombres de arreglo son

```
test1year1
test2year1
test3year1
test1year2
test2year2
test3year2
```

Organice estos datos en un arreglo tetradimensional que se parezca al siguiente:

dimensión 1	(fila)	estudiante
dimensión 2	(columna)	pregunta
dimensión 3	(página)	año
dimensión 4	(drawer	prueba

(a) Extraiga la calificación para el Estudiante 1, en la Pregunta 2, del primer año, en la Prueba 3.
(b) Cree un arreglo unidimensional que represente las calificaciones para el primer estudiante, en la Pregunta 1, en la segunda prueba, para todos los años.
(c) Cree un arreglo unidimensional que represente las calificaciones del segundo estudiante, en todas las preguntas, en la primera prueba, para el Año 2.
(d) Cree un arreglo bidimensional que represente las calificaciones de todos los estudiantes, en la Pregunta 3, de la segunda prueba, para todos los años.

Arreglos carácter

10.10 **(a)** Cree un arreglo carácter acolchado con cinco nombres diferentes.
(b) Cree un arreglo bidimensional llamado **birthdays** para representar el cumpleaños de cada persona. Por ejemplo, su arreglo puede parecer algo como esto:

```
birthdays =
      6      11     1983
      3      11     1985
      6      29     1986
      12     12     1984
      12     11     1987
```

(c) Use la función **num2str** para convertir **birthdays** a un arreglo carácter.
(d) Use la función **disp** para desplegar una tabla de nombres y cumpleaños.

10.11 Imagine que tiene el siguiente arreglo carácter que representa las dimensiones de algunas cajas de embarque:

```
box_dimensions =

box1 1  3  5
box2 2  4  6
box3 6  7  3
box4 1  4  3
```

Necesita encontrar los volúmenes de las cajas para usar en un cálculo que determine cuántos "cacahuates" empacados ordenar para su departamento de embarque. Dado que el arreglo es un arreglo carácter 4×12, la representación carácter de la información numérica se almacena en las columnas de la 6 a la 12. Use la función **str2num** para convertir la información en un arreglo numérico y use los datos para calcular el volumen de cada caja. (Necesitará ingresar el arreglo **box_dimensions** como datos cadena, con la función **char**.)

10.12 Considere el siguiente archivo llamado **thermocouple.dat**:

Termocople 1	Termocople 2	Termocople 3
84.3	90.0	86.7
86.4	89.5	87.6
85.2	88.6	88.3
87.1	88.9	85.3
83.5	88.9	80.3
84.8	90.4	82.4
85.0	89.3	83.4
85.3	89.5	85.4
85.3	88.9	86.3
85.2	89.1	85.3
82.3	89.5	89.0
84.7	89.4	87.3
83.6	89.8	87.2

(a) Cree un programa que

- cargue **thermocouple.dat** en MATLAB.
- determine el tamaño (número de filas y columnas) del archivo.
- extraiga cada conjunto de datos termocople y los almacene en un archivo separado. Nombre los diversos archivos **thermocouple1.mat**, **thermocouple2.mat**, etcétera.

(b) Su programa debe ser capaz de aceptar cualquier tamaño de archivo bidimensional. No suponga que sólo existen tres columnas; haga que el programa determine el tamaño del arreglo y asigne nombres de archivo apropiados.

10.13 Cree un programa que codifique el texto ingresado por el usuario y lo guarde en un archivo. Su código debe agregar 10 al valor equivalente decimal de cada carácter ingresado.

10.14 Cree un programa para decodificar un mensaje almacenado en un archivo de datos al restar 10 del valor equivalente decimal de cada carácter.

Arreglos celda

10.15 Cree un arreglo celda llamado **sample_cell** para almacenar los siguientes arreglos individuales:

$$A = \begin{bmatrix} 1 & 3 & 5 \\ 3 & 9 & 2 \\ 11 & 8 & 2 \end{bmatrix} \quad \text{(arreglo punto flotante precisión doble)}$$

$$B = \begin{bmatrix} \mathit{fred} & \mathit{ralph} \\ \mathit{ken} & \mathit{susan} \end{bmatrix} \quad \text{(arreglo carácter acolchado)}$$

$$C = \begin{bmatrix} 4 \\ 6 \\ 3 \\ 1 \end{bmatrix} \quad \text{(arreglo entero } \textbf{int8}\text{)}$$

(a) Extraiga el arreglo A de **sample_cell**.
(b) Extraiga la información en el arreglo C, fila 3, de **sample:cell**.
(c) Extraiga el nombre *fred* de **sample_cell**. Recuerde que el nombre **fred** es un arreglo 1×4, no una entidad sola.

10.16 Los arreglos celda se pueden usar para almacenar información carácter sin acolchar los arreglos carácter. Cree un arreglo carácter separado para cada una de las cadenas

aluminio
cobre
hierro
molibdeno
cobalto

y almacénelos en un arreglo celda.

10.17 Considere la siguiente información acerca de los metales:

Metal	Símbolo	Número atómico	Peso atómico	Densidad, g/cm^3	Estructura cristalina
Aluminio	Al	13	26.98	2.71	FCC
Cobre	Cu	29	63.55	8.94	FCC
Hierro	Fe	26	55.85	7.87	BCC
Molibdeno	Mo	42	95.94	10.22	BCC
Cobalto	Co	27	58.93	8.9	HCP

(a) Cree los siguientes arreglos:

- Almacene el nombre de cada metal en un arreglo carácter individual y almacene todos estos arreglos carácter en un arreglo celda.
- Almacene el símbolo de todos estos metales en un solo arreglo carácter acolchado.
- Almacene el número atómico en un arreglo entero **int8**.
- Almacene el peso atómico en un arreglo numérico de precisión doble.
- Almacene la densidad en un solo arreglo numérico de precisión doble.
- Almacene la estructura en un solo arreglo carácter acolchado.

(b) Agrupe los arreglos que creó en la parte (a) en un arreglo celda de precisión sencilla.

(c) Extraiga la siguiente información de su arreglo celda:

- Encuentre el nombre, peso atómico y estructura del cuarto elemento en la lista.
- Encuentre el nombre de todos los elementos almacenados en el arreglo.
- Encuentre el peso atómico promedio de los elementos en la tabla. (Recuerde, necesita extraer la información que usó en su cálculo del arreglo celda.)

Arreglos estructura

10.18 Almacene la información presentada en el problema 10.17 en un arreglo estructura. Use su arreglo estructura para determinar el elemento con la máxima densidad.

10.19 Cree un programa que permita al usuario ingresar información adicional en el arreglo estructura que creó en el problema 10.18. Use su programa para agregar los siguientes datos al arreglo:

Metal	Símbolo	Número atómico	Peso atómico	Densidad, g/cm^3	Estructura cristalina
Litio	Li	3	6.94	0.534	BCC
Germanio	Ge	32	72.59	5.32	diamante cúbico
Oro	Au	79	196.97	19.32	FCC

10.20 Use el arreglo estructura que creó en el problema 10.19 para encontrar el elemento con el máximo peso atómico.



CAPÍTULO

11

Matemática simbólica

Objetivos

Después de leer este capítulo, el alumno será capaz de

- crear y manipular variables simbólicas.
- factorizar y simplificar expresiones matemáticas.
- resolver expresiones simbólicas.
- resolver sistemas de ecuaciones.
- determinar la derivada simbólica de una expresión.
- integrar una expresión.

INTRODUCCIÓN

MATLAB tiene algunos tipos diferentes de datos, incluidos datos numéricos de precisión doble y sencilla, datos carácter, datos lógicos y datos simbólicos, los cuales se almacenan en una variedad de distintos arreglos. En este capítulo se explorará cómo los arreglos simbólicos permiten a los usuarios de MATLAB manipular y usar datos simbólicos.

La capacidad simbólica de MATLAB se basa en el software Maple 8, producido por Waterloo Maple. (Ésta es una actualización en MATLAB 7; las versiones anteriores usaban Maple 5.) El motor Maple 8 es parte de la caja de herramientas simbólica. Si usted ha usado Maple anteriormente, encontrará similares la mayoría de los comandos simbólicos MATLAB. Sin embargo, puesto que a diferente de Maple, MATLAB usa el arreglo como su tipo de datos principal, y puesto que el motor Maple está incrustado en MATLAB, la sintaxis se modificó para ser consistente con las convenciones y capacidades subyacentes a MATLAB.

La caja de herramientas simbólica es una característica opcional de la versión profesional de MATLAB 7 y se debe instalar con la finalidad de que funcionen los ejemplos que siguen. Con la edición estudiantil de MATLAB se incluye un subconjunto de la caja de herramientas simbólica de MATLAB 7, porque se usa ampliamente. Si usted tiene una versión anterior de MATLAB (Release 12 o anterior), es posible que no funcionen algunos de los ejercicios descritos en este capítulo.

La caja de herramientas simbólicas de MATLAB le permite manipular expresiones simbólicas para simplificarlas, resolverlas simbólicamente y evaluarlas numéricamente. También le permite obtener derivadas, integrar y realizar manipulaciones algebraicas lineales. Las características más avanzadas incluyen transformadas de LaPlace, transformadas de Fourier y aritmética de precisión variable.

11.1 ÁLGEBRA SIMBÓLICA

La matemática simbólica se usa regularmente en las clases de matemáticas, ingeniería y ciencias. Con frecuencia es preferible manipular las ecuaciones simbólicamente antes de sustituir valores para las variables. Por ejemplo, considere la ecuación

Idea clave: la caja de herramientas simbólica es un componente opcional de la versión profesional, pero es estándar con la versión estudiantil.

$$y = \frac{2(x + 3)^2}{x^2 + 6x + 9}$$

Cuando la observa por primera vez, parece que y es una función bastante complicada de x. Sin embargo, si expande la cantidad $(x + 3)^2$, se hace evidente que puede simplificar la ecuación a

$$y = \frac{2*(x + 3)^2}{x^2 + 6x + 9} = \frac{2*(x^2 + 6x + 9)}{(x^2 + 6x + 9)} = 2$$

Acaso quiera o no realizar esta simplificación, porque al hacerla pierde algo de información. Por ejemplo, para valores de x igual a –3, y está indefinida, pues $x + 3$ se vuelve 0, al igual que $x^2 + 6x + 9$. Por tanto,

$$y = \frac{2(-3 + 3)^2}{9 - 18 + 9} = 2\frac{0}{0} = \text{indefinido}$$

Las capacidades simbólicas de MATLAB le permiten realizar esta simplificación o manipular el numerador y denominador por separado.

Las relaciones no siempre se constituyen en formas fáciles de resolver. Por ejemplo, considere la ecuación

$$D = D_0 e^{-Q/RT}$$

Idea clave: MATLAB facilita la resolución de ecuaciones simbólicamente.

Si se conocen los valores de D_0, Q, R y T, es fácil resolver para D. No es tan fácil si se quiere encontrar T y se conocen los valores de D, D_0, R y Q. Tiene que manipular la relación para obtener T en el lado izquierdo de la ecuación:

$$\ln(D) = \ln(D_0) - \frac{Q}{RT}$$

$$\ln\left(\frac{D}{D_0}\right) = -\frac{Q}{RT}$$

$$\ln\left(\frac{D_0}{D}\right) = \frac{Q}{RT}$$

$$T = \frac{Q}{R\ln(D_0/D)}$$

Aunque resolver para T es complicado manualmente, es fácil con las capacidades simbólicas de MATLAB.

11.1.1 Creación de variables simbólicas

Las variables simbólicas simples se pueden crear en dos formas. Por ejemplo, para crear la variable simbólica **x**, escriba o

```
x=sym('x')
```

o

```
syms x
```

Ambas técnicas hacen al carácter **'x'** igual a la variable simbólica **x**. Se pueden crear variables más complicadas usando las variables simbólicas existentes, como en la expresión

```
y = 2*(x+3)^2/(x^2+6*x+9)
```

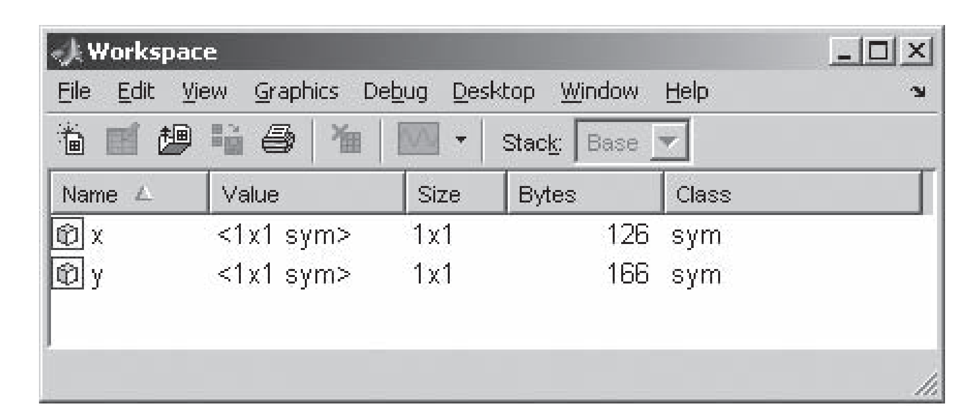


Figura 11.1
Las variables simbólicas se identifican en la ventana del área de trabajo. Requieren una cantidad de almacenamiento variable.

Note en la ventana del área de trabajo (figura 11.1) que **x** y **y** se mencionan como variables simbólicas y que el tamaño del arreglo para cada una es 1 × 1.

Idea clave: las expresiones son diferentes de las ecuaciones.

El comando **syms** es particularmente conveniente porque se puede usar para crear múltiples variables simbólicas al mismo tiempo, como con el comando

```
syms Q R T D0
```

Estas variables se pueden combinar matemáticamente para crear otra variable simbólica, **D**:

expresión: conjunto de operaciones matemáticas

```
D=D0*exp(-Q/(R*T))
```

Note que en ambos ejemplos se usaron los operadores algebraicos estándar, no los operadores de arreglo, como **.*** o **.^**. Esto tiene sentido cuando se observa que los operadores de arreglo especifican que elementos correspondientes en arreglos se usan en los cálculos asociados, una situación que no se aplica aquí.

La función **sym** también se puede usar para crear una expresión entera o una ecuación entera. Por ejemplo,

ecuación: expresión que se iguala a un valor u otra expresión

```
E=sym('m*c^2')
```

crea una variable simbólica llamada **E**. Note que **m** y **c** no se mencionan en la ventana del área de trabajo (figura 11.2); no se tienen que definir específicamente como variables simbólicas. En vez de ello, **E** se igualó a una cadena carácter, definida por el apóstrofe dentro de la función.

En este ejemplo, la *expresión* **m*c^2** se iguala con la variable **E**. También se puede crear una *ecuación* entera y darle un nombre. Por ejemplo, se puede definir la ley del gas ideal

Idea clave: la caja de herramientas simbólica usa operadores algebraicos estándar.

```
ideal_gas_law=sym('P*V=n*R*Temp')
```

En este punto, si ha escrito los ejemplos conforme los lee, su ventana de área de trabajo debe parecerse a la de la figura 11.3. Note que sólo **ideal_gas_law** se menciona como variable simbólica, pues **P**, **V**, **n**, **R** y **Temp** no se definieron explícitamente, sino que fueron parte de la cadena carácter de entrada a la función **sym**.

Workspace

File Edit View Graphics Debug Desktop Window Help

Stack: Base

Name	Value	Size	Bytes	Class
D	<1x1 sym>	1x1	152	sym
D0	<1x1 sym>	1x1	128	sym
E	<1x1 sym>	1x1	134	sym
Q	<1x1 sym>	1x1	126	sym
R	<1x1 sym>	1x1	126	sym
T	<1x1 sym>	1x1	126	sym
x	<1x1 sym>	1x1	126	sym
y	<1x1 sym>	1x1	166	sym

Figura 11.2
A menos que una variable se defina explícitamente, no se cita en la ventana del área de trabajo.

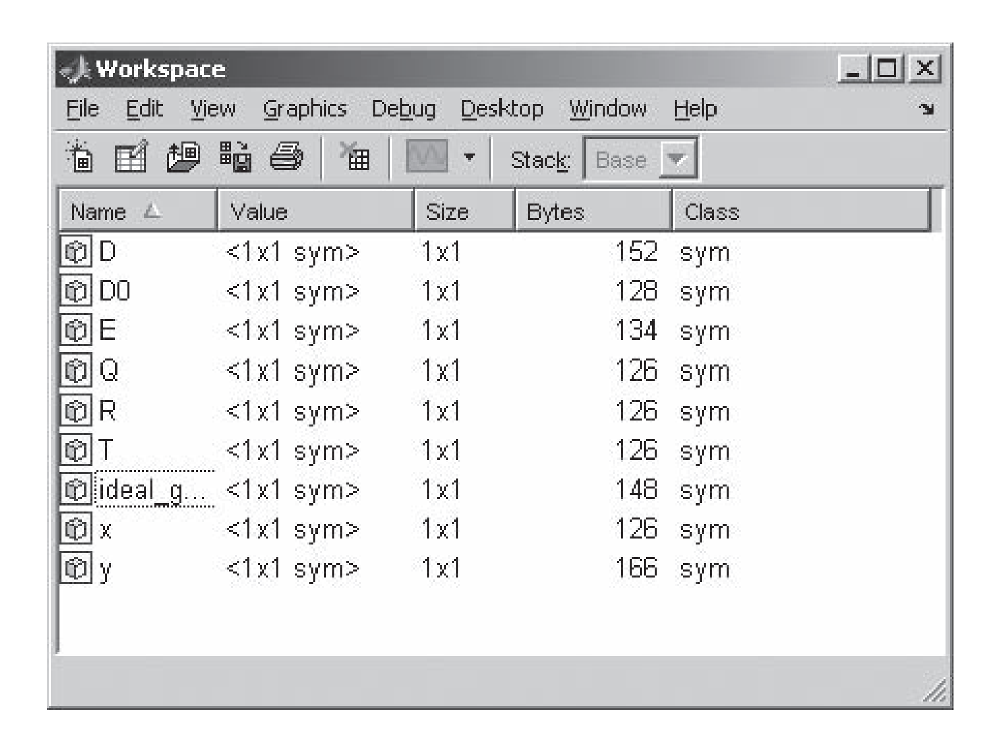


Figura 11.3
La variable **ideal_gas_law** es un ecuación, no una expresión.

Ejercicio de práctica 11.1

1. Cree las siguientes variables simbólicas con el comando **sym** o **syms**:

```
x, a, b, c, d
```

2. Verifique que las variables que creó en el problema 1 se mencionan en la ventana del área de trabajo como variables simbólicas. Úselas para crear las siguientes *expresiones* simbólicas:

```
ex1 = x^2-1
ex2 = (x+1)^2
ex3 = a*x^2-1
ex4 = a*x^2 + b*x + c
ex5 = a*x^3 + b*x^2 + c*x + d
ex6 = sin(x)
```

3. Cree las siguientes *expresiones* simbólicas, con la función **sym**:

```
EX1 = sym('X^2 - 1 ')
EX2 = sym(' (X +1)^2 ')
EX3 = sym('A*X ^2 - 1 ')
EX4 = sym('A*X ^2 + B*X + C ')
EX5 = sym('A*X ^3 + B*X ^2 + C*X + D ')
EX6 = sym('sin(X) ')
```

4. Cree las siguientes *ecuaciones* simbólicas, con la función **sym**:

```
eq1 = sym(' x^2=1 ')
eq2 = sym(' (x+1)^2=0 ')
eq3 = sym(' a*x^2=1 ')
eq4 = sym('a*x^2 + b*x + c=0 ')
eq5 = sym('a*x^3 + b*x^2 + c*x + d=0 ')
eq6 = sym('sin(x)=0 ')
```

5. Cree las siguientes *ecuaciones* simbólicas, con la función **sym**:

```
EQ1 = sym('X^2 = 1 ')
EQ2 = sym('(X +1)^2=0 ')
EQ3 = sym('A*X ^2 =1 ')
EQ4 = sym('A*X ^2 + B*X + C = 0 ')
EQ5 = sym('A*X ^3 + B*X ^2 + C*X + D = 0 ')
EQ6 = sym(' sin(X) = 0 ')
```

Note que sólo las variables, expresiones y ecuaciones definidas explícitamente se mencionan en la ventana del área de trabajo. Guarde las variables, expresiones y ecuaciones que cree en esta práctica para usarlas más tarde en los ejercicios de práctica del capítulo.

11.1.2 Manipulación de expresiones y ecuaciones simbólicas

Primero debe recordar cómo difieren las expresiones y ecuaciones. Las ecuaciones se igualan a algo; las expresiones no. La variable **ideal_gas_law** se igualó a una ecuación. Si escribe

```
ideal_gas_law
```

MATLAB responderá

```
ideal_gas_law =
P*V=n*R*Temp
```

Sin embargo, si escribe

```
E
```

MATLAB responde

```
E=
m*c^2
```

o si escribe

```
y
```

MATLAB responde

```
y =
2*(x+3)^2/(x^2+6*x+9)
```

Las variables **E** y **y** son *expresiones*, pero la variable **ideal_gas_law** es una *ecuación*. La mayor parte del tiempo trabajará con *expresiones* simbólicas.

Sugerencia

Note que cuando usa variables simbólicas MATLAB no sangra el resultado, a diferencia del formato que se usa para resultados numéricos. Esto puede ayudarlo a rastrear los tipos de variable sin hacer referencia a la ventana del área de trabajo.

MATLAB tiene algunas funciones diseñadas para manipular variables simbólicas, incluidas funciones para separar una expresión en su numerador y denominador, para expandir o factorizar expresiones, y algunas formas de simplificar expresiones.

Extracción de numeradores y denominadores

La función **numden** extrae el numerador y el denominador de una expresión. Por ejemplo, si definió **y** como

```
y = 2*(x+3)^2/(x^2+6*x+9)
```

entonces puede extraer el numerador y denominador con

```
[num,den] = numden(y)
```

MATLAB crea dos nuevas variables, **num** y **den** (desde luego, podría llamarlas como a usted le guste):

```
num =
2*(x+3)^2
den =
x^2+6*x+9
```

Estas o cualesquiera expresiones simbólicas se pueden recombinar al usar operadores algebraicos estándar:

```
num*den
ans =
2*(x+3)^2*(x^2+6*x+9)

num/den
ans =
2*(x+3)^2/(x^2+6*x+9)
```

```
num+den
ans =
2*(x+3)^2+x^2+6*x+9
```

Expansión de expresiones, factorización de expresiones y recolección de términos

Se pueden usar las expresiones definidas para demostrar el uso de las funciones **expand**, **factor** y **collect**. Por tanto,

```
expand(num)
```

regresa

```
ans =
2*x^2+12*x+18
```

y

```
factor(den)
```

regresa

```
ans =
(x+3)^2
```

La función **collect** recopila términos iguales y es similar a la función **expand**:

```
collect(num)
ans =
18+2*x^2+12*x
```

Esto funciona sin importar si cada variable individual en una expresión fue o no definida como variable simbólica. Defina una nueva variable **z**:

```
z = sym('3*a-(a+3)*(a-3)^2')
```

En este caso, tanto **expand** como **factor** dan el mismo resultado:

```
factor(z)

ans =
12*a-a^3+3*a^2-27
expand(z)
ans =
12*a-a^3+3*a^2-27
```

El resultado que se obtuvo al usar **collect** es similar; la única diferencia es el orden en el que se mencionan los términos:

```
collect(z)
ans =
-27-a^3+3*a^2+12*a
```

Puede usar las tres funciones con ecuaciones así como con expresiones. Con las ecuaciones, cada lado de la ecuación se trata como una expresión separada. Para ilustrar, se puede definir una ecuación **w**:

```
w=sym('x^3-1=(x-3)*(x+3)')

expand(w)
ans =
```

```
x^3-1 = x^2-9

factor(w)
ans =
(x-1)*(x^2+x+1) = (x-3)*(x+3)

collect(w)
ans =
x^3-1 = x^2-9
```

Simplificación de funciones

Se puede pensar en las funciones **expand**, **factor** y **collect** como formas de simplificar una ecuación. Sin embargo, qué constituye una ecuación "simple" no siempre es obvio. La función **simplify** simplifica cada parte de una expresión o ecuación, al usar las reglas de simplificación internas de Maple. Por ejemplo, suponga de nuevo que **z** se definió como

```
z=sym('3*a-(a+3)*(a-3)^2')
```

Entonces el comando

```
simplify(z)
```

regresa

```
ans =
12*a-a^3+3*a^2-27
```

Si la ecuación **w** se definió como

```
w=sym('x^3-1=(x-3)*(x+3)')
```

entonces

```
simplify(w)
```

regresa

```
ans =
x^3-1 = x^2-9
```

Note de nuevo que esto funciona sin importar si cada variable individual en una expresión fue o no definida como una variable simbólica: la expresión **z** contiene la variable **a**, que no se definió explícitamente y que no se menciona en la ventana del área de trabajo.

La función **simple** es ligeramente diferente. Intente algunas técnicas de simplificación diferentes y reporte el resultado que es el *más corto*. Todos los intentos se reportan en la pantalla. Por ejemplo,

```
simple(w)
```

da los siguientes resultados:

```
simplify:
x^3-1 = x^2-9
radsimp:
x^3-1 = (x-3)*(x+3)
```

```
combine(trig):
x^3-1 = x^2-9
factor:
(x-1)*(x^2+x+1) = (x-3)*(x+3)
expand:
x^3-1 = x^2-9
combine:
x^3-1 = (x-3)*(x+3)
convert(exp):
x^3-1 = (x-3)*(x+3)
convert(sincos):
x^3-1 = (x-3)*(x+3)
convert(tan):
x^3-1 = (x-3)*(x+3)
collect(x):
x^3-1 = x^2-9
mwcos2sin:
x^3-1 = (x-3)*(x+3)
ans =
x^3-1 = x^2-9
```

Idea clave: MATLAB define la representación más simple de una expresión como la versión más corta de la expresión.

Idea clave: muchas, mas no todas las expresiones simbólicas funcionan tanto para expresiones como para ecuaciones.

Note que, aunque una gran cantidad de resultados se despliegan en la pantalla, sólo hay una respuesta:

```
ans =
x^2-1 = x^2-9
```

Tanto **simple** como **simplify** funcionan en expresiones lo mismo que en ecuaciones.

La tabla 11.1 menciona algunas de las funciones MATLAB que se usan para manipular expresiones y ecuaciones.

Sugerencia

Un atajo para crear un polinomio simbólico es la función **poly2sym**. Esta función requiere un vector como entrada y crea un polinomio, usando el vector para los coeficientes de cada término del polinomio.

```
a=[1,3,2]
a =
   1    3    2
b=poly2sym(a)
b =
x^2+3*x+2
```

De modo similar, la función **sym2poly** convierte un polinomio en un vector de valores coeficiente:

```
c=sym2poly(b)
c =
     1    3    2
```

Tabla 11.1 Funciones que se usan para manipular expresiones y ecuaciones

`expand(S)`	multiplica todas las porciones de la expresión o ecuación	`syms x`
		`expand((x-5)*(x+5))`
		`ans =`
		`x^2-25`

`factor(S)`	factoriza la expresión o ecuación	`syms x`
		`factor(x^3-1)`
		`ans =`
		`(x-1)*(x^2+x+1)`

`collect(S)`	recopila términos similares	`S=2*(x+3)^2+x^2+6*x+9`
		`collect(S)`
		`S =`
		`27+3*x^2+18*x`

`simplify(S)`	simplifica en concordancia con las reglas de simplificación de Maple	`syms a`
		`simplify(exp(log(a)))`
		`ans =`
		`a`

`simple(S)`	simplifica a la representación más corta de la expresión o ecuación	`syms x`
		`simple(sin(x)^2+`
		`cos(x)^2)`
		`ans=1`
		`1`

`numden(S)`	encuentra el numerador de una expresión; esta función no es válida para ecuaciones	`syms x`
		`numden((x-5)/(x+5))`
		`ans =`
		`x-5`

`[num,den]=numden(S)`	encuentra tanto el numerador como el denominador de una expresión; esta función no es válida para ecuaciones	`syms x`
		`[num,den]=numden((x-`
		`5)/(x+5))`
		`num =`
		`x-5`
		`den =`
		`x+5`

Ejercicio de práctica 11.2

Use en estos ejercicios las variables definidas en la práctica 11.1.

1. Multiplique **ex1** por **ex2** y llame al resultado **y1**.
2. Divida **ex1** entre **ex2** y llame al resultado **y2**.
3. Use la función **numden** para extraer el numerador y denominador de **y1** y **y2**.
4. Multiplique **EX1** por **EX2** y llame al resultado **Y1**.
5. Divida **EX1** entre **EX2** y llame al resultado **Y2**.
6. Use la función **numden** para extraer el numerador y denominador de **Y1** y **Y2**.
7. Intente usar la función **numden** en una de las ecuaciones que definió. ¿Funciona?
8. Use las funciones **factor**, **expand**, **collect** y **simplify** en **y1**, **y2**, **Y1** y **Y2**.

9. Use las funciones **factor**, **expand**, **collect** y **simplify** en las expresiones **ex1** y **ex2** y sobre las correspondientes ecuaciones **eq1** y **eq2**. Explique las diferencias que observe.

11.2 RESOLUCIÓN DE EXPRESIONES Y ECUACIONES

Una de las funciones más útiles en la caja de herramientas simbólica es **solve**. Se le puede usar para determinar las raíces de expresiones, para encontrar respuestas numéricas cuando hay una sola variable y para resolver simbólicamente una incógnita. La función **solve** también puede resolver sistemas de ecuaciones tanto lineales como no lineales. Cuando se parea con la función sustitución (**subs**), la función **solve** permite al usuario encontrar soluciones analíticas a una variedad de problemas.

11.2.1 La función solve

Cuando se usa con una expresión, la función **solve** iguala la expresión a cero y resuelve para las raíces. Por ejemplo (si supone que **x** ya se definió como una variable simbólica), si

```
E1=x-3
```

entonces

```
solve(E1)
```

regresa

```
ans =
3
```

Idea clave: MATLAB resuelve preferentemente para x.

Solve se puede usar con el nombre de una expresión o creando una expresión simbólica directamente en la función **solve**. Por tanto,

```
solve('x^2-9')
```

regresa

```
ans =
3
-3
```

Note que **ans** es un arreglo simbólico 2 × 1. Si **x** se definió anteriormente como una variable simbólica, entonces *los apóstrofes no son necesarios*. Si no, toda la expresión se debe encerrar dentro de apóstrofes.

Usted puede resolver fácilmente expresiones simbólicas con más de una variable. Por ejemplo, para la ecuación cuadrática $ax^2 + bx + c$,

```
solve('a*x^2+b*x +c')
```

regresa

```
ans =
1/2/a*(-b+(b^2-4*a*c)^(1/2))
1/2/a*(-b-(b^2-4*a*c)^(1/2))
```

MATLAB resuelve preferentemente para **x**. Si no hay **x** en la expresión, MATLAB encuentra la variable más cercana a **x**. Si quiere especificar la variable por resolver, sólo in-

clúyala en el segundo campo. Por ejemplo, para resolver la ecuación cuadrática para **a**, el comando

```
solve('a*x^2+b*x +c', 'a')
```

regresa

```
ans =
-(b*x+c)/x^2
```

De nuevo, si **a** se definió específicamente como variable simbólica, no es necesario encerrarla entre apóstrofes:

```
syms a b c x
solve(a*x*x^2+b*x+c,b)
ans =
-(a*x^3+c)/x
```

Para resolver una expresión igualada a algo, además de cero requiere que use uno de dos enfoques. Si la ecuación es simple, puede transformarla en una expresión al restar el lado derecho del lado izquierdo. Por ejemplo,

$$5x^2 + 6x + 3 = 10$$

se podría reformular como

$$5x^2 + 6x - 7 = 0$$

Idea clave: incluso cuando el resultado de la función **solve** sea un número, todavía se almacena como una variable simbólica.

```
solve('5*x^2+6*x-7')
ans =
-3/5+2/5*11^(1/2)
-3/5-2/5*11^(1/2)
```

Si la ecuación es más complicada, debe definir una nueva ecuación, como en

```
E2=sym('5*x^2 + 6*x +3=10')
solve(E2)
```

que regresa

```
ans =
-3/5+2/5*11^(1/2)
-3/5-2/5*11^(1/2)
```

Note que, en ambos casos, los resultados se expresan tan simplemente como sea posible usando fracciones (es decir: números racionales). En el área de trabajo, **ans** se menciona como matriz simbólica 2×1. Puede usar la función **double** para convertir una representación simbólica a un número punto flotante de precisión doble:

```
double(ans)
ans =
 0.7266
-1.9266
```

Sugerencia

Puesto que las capacidades simbólicas de MATLAB se basan en Maple, necesita entender cómo Maple maneja los cálculos. Maple reconoce dos tipos de datos numéricos: enteros y punto flotante. Los números punto flotante se consideran aproximaciones y usan punto decimal, mientras que los enteros son exactos y se representan sin punto decimal. En los cálculos que usan enteros, Maple fuerza una respuesta exacta, que resulta en fracciones. Si hay punto decimal (números punto flotante) en los cálculos Maple, el resultado también será una aproximación y contendrá punto decimal. Maple tiene por defecto 32 cifras significativas, de modo que en los resultados se muestran 32 dígitos. Considere un ejemplo que usa **solve**. Si la expresión usa números punto flotante, se obtiene el siguiente resultado:

```
solve('5.0*x^2.0+6.0*x-7.0')
ans =
.72664991614215993964597309466828
-1.9266499161421599396459730946683
```

Si la expresión usa enteros, los resultados son fracciones:

```
solve('5*x^2+6*x-7')
ans =
-3/5+2/5*11^(1/2)
-3/5-2/5*11^(1/2)
```

La función **solve** es particularmente útil con expresiones simbólicas con múltiples variables:

```
E3=sym('P=P0*exp(r*t)')
solve(E3,'t')

ans =
log(P/P0)/r
```

Si anteriormente definió **t** como variable simbólica, no necesita apóstrofes. (Recuerde que la función log es un logaritmo natural.)

Con frecuencia es útil redefinir una variable, como **t**, en términos de las otras variables:

```
t=solve(E3,'t')
t =
log(P/P0)/r
```

Ejercicio de práctica 11.3

Use las variables y expresiones que definió en la práctica 11.1 para resolver estos ejercicios:

1. Use la función **solve** para resolver las cuatro versiones de expresión/ecuación 1: **ex1**, **EX1**, **eq1** y **EQ1**.
2. Use la función **solve** para resolver las cuatro versiones de expresión/ecuación 2: **ex2**, **EX2**, **eq2** y **EQ2**.

3. Use la función **solve** para resolver **ex3** y **eq3** tanto para **x** como para **a**.
4. Use la función **solve** para resolver **EX3** y **EQ3** tanto para **X** como para **A**. Recuerde que ni **X** ni **A** se definieron explícitamente como variable simbólica.
5. Use la función **solve** para resolver **ex4** y **eq4** tanto para **x** como para **a**.
6. Use la función **solve** para resolver **EX4** y **EQ4** tanto para **X** como para **A**. Recuerde que ni **X** ni **A** se definieron explícitamente como variable simbólica.
7. Las cuatro versiones de expresión/ecuación 4 representan la ecuación cuadrática, la forma general de un polinomio de segundo orden. La solución de esta ecuación para x usualmente la memorizan los estudiantes en las primeras clases de álgebra. La expresión/ecuación 5 en estos ejercicios es la forma general de un polinomio de tercer orden. Use la función **solve** para resolver estas expresiones/ecuaciones y comente por qué los estudiantes no memorizan la solución general de un polinomio de tercer orden.
8. Use la función **solve** para resolver **ex6**, **EX6**, **eq6** y **EQ6**. Sobre la base de su conocimiento de trigonometría, comente esta solución.

EJEMPLO 11.1

Uso de matemática simbólica

Las capacidades simbólicas de MATLAB le permiten hacer que la computadora haga las matemáticas. Considere la ecuación para difusividad:

$$D = D_0 \exp\left(\frac{-Q}{RT}\right)$$

Resuelva esta ecuación para Q usando MATLAB.

1. Establezca el problema.
   Encontrar la ecuación para Q.
2. Describa las entradas y salidas.

   Entrada Ecuación para D

   Salida Ecuación para Q
3. Desarrolle un ejemplo a mano.

$$D = D_0 \exp\left(\frac{-Q}{RT}\right)$$

$$\frac{D}{D_0} = \exp\left(\frac{-Q}{RT}\right)$$

$$\ln\left(\frac{D}{D_0}\right) = \frac{-Q}{RT}$$

$$Q = RT \ln\left(\frac{D_0}{D}\right)$$

   Note que el signo menos hizo que se inviertan los valores dentro del logaritmo natural.
4. Desarrolle una solución MATLAB.
   Defina primero una ecuación simbólica y asígnele un nombre (recuerde que está bien colocar una igualdad dentro de la expresión):

```
X = sym('D = D0*exp(-Q/(R*T))')
X =
D = D0*exp(-Q/(R*T))
```

Ahora pida a MATLAB resolver la ecuación. Es necesario aclarar que MATLAB resolverá para *Q*, y que *Q* necesita estar entre apóstrofes, porque no se definió por separado como variable simbólica:

```
solve(X,'Q')
ans =
-log(D/D0)*R*T
```

De manera alternativa, se podría definir la respuesta como **Q**:

```
Q = solve(X, 'Q')
Q =
-log(D/D0)*R*T
```

5. Ponga a prueba la solución.
   Compare la solución MATLAB con la solución a mano. La única diferencia es que se empujó el signo menos dentro del logaritmo. Note que MATLAB (así como la mayoría de los programas de cómputo) representa ln como **log** ($\log_{10}$ se representa como **log10**).

   Ahora que se sabe que esta estrategia funciona, se podría resolver para cualquiera de las variables. Por ejemplo, se podría tener

```
T=solve(X,'T')
T =
-Q/log(D/D0)/R
```

Sugerencia

El comando **findsym** es útil para determinar cuáles variables existen en una expresión o ecuación simbólica. En el ejemplo previo, la variable *X* se definió como

```
X = sym('D = D0*exp(-Q/(R*T))')
```

La función **findsym** identifica todas las variables, ya sea que se definieron explícitamente o no:

```
findsym(X)
ans =
D, D0, Q, R, T
```

11.2.2 Resolución de sistemas de ecuaciones

La función **solve** no sólo puede resolver ecuaciones o expresiones sencillas para algunas de las variables incluidas, también puede resolver sistemas de ecuaciones. Tome, por ejemplo, estas tres ecuaciones simbólicas:

```
one = sym('3*x + 2*y -z = 10');
two = sym('-x + 3*y + 2*z = 5');
three = sym('x - y - z = -1');
```

Para resolver las tres variables incrustadas **x**, **y** y **z**, simplemente mencione las tres ecuaciones en la función **solve**:

Idea clave: la función **solve** puede resolver sistemas de ecuaciones tanto lineales como no lineales.

```
answer=solve(one,two,three)
answer =
  x: [1x1 sym]
  y: [1x1 sym]
  z: [1x1 sym]
```

Estos resultados son intrigantes. Cada respuesta se menciona como una variable simbólica 1 × 1, pero el programa no revela los valores de dichas variables. Además, **answer** se menciona en la ventana del área de trabajo como un arreglo estructura 1 × 1. Para acceder a los valores reales, necesitará usar la sintaxis del arreglo estructura:

Idea clave: los resultados de la función **solve** simbólica se mencionan alfabéticamente.

```
answer.x
ans =
-2
answer.y
ans =
5
answer.z
ans =
-6
```

Para forzar los resultados a desplegarse sin usar un arreglo estructura y la sintaxis asociada, debe asignar nombres a las variables individuales. Por tanto, para el ejemplo, se tiene

```
[x,y,z]=solve(one,two,three)
x =
-2
y =
5
z =
-6
```

Los resultados se asignan alfabéticamente. Por ejemplo, si las variables que usó en sus expresiones simbólicas son **q**, **x** y **p**, los resultados se regresarán en el orden **p**, **q**, **x**, *independientemente* de los nombres que asignó para los resultados.

Note en el ejemplo que **x**, **y** y **z** todavía se mencionan como variables simbólicas, aun cuando los resultados sean números. El resultado de la función **solve** es una variable simbólica, o **ans** o un nombre definido por el usuario. Si quiere usar dicho resultado en una expresión MATLAB que requiere una entrada punto flotante de precisión doble, puede cambiar el tipo de variable con la función **double**. Por ejemplo,

```
double(x)
```

cambia **x** de una variable simbólica a una variable numérica correspondiente.

Sugerencia

Usar la función **solve** para ecuaciones múltiples tiene ventajas y desventajas sobre el uso de las técnicas del álgebra lineal. En general, si un problema se puede resolver mediante matrices, la solución matricial tomará menos tiempo de cómputo. Sin embargo, el álgebra lineal está limitada a ecuaciones de primer orden. La función **solve** puede tardar más, pero puede resolver problemas no lineales y problemas con variables simbólicas. La tabla 11.2 menciona algunos usos de la función **solve**.

Tabla 11.2 Uso de la función solve

`solve(S)`	resuelve una expresión con una sola variable	`solve('x-5')`
		`ans =`
		`5`

`solve(S)`	resuelve una ecuación con una sola variable	`solve('x^2-2=5')`
		`ans =`
		` 7^(1/2)`
		`-7^(1/2)`

`solve(S)`	resuelve una ecuación cuyas soluciones son números complejos	`solve('x^2=-5')`
		`ans =`
		` i*5^(1/2)`
		`-i*5^(1/2)`

`solve(S)`	resuelve una ecuación con más de una variable para *x* o la variable más cercana a *x*	`solve('y=x^2+2')`
		`ans =`
		`  (y-2)^(1/2)`
		` -(y-2)^(1/2)`

`solve(S,y)`	resuelve una ecuación con más de una variable para una variable especificada	`solve('y+6*x')`
		`ans =`
		`-1/6*y`

`solve(S1,S2,S3)`	resuelve un sistema de ecuaciones y presenta las soluciones como un arreglo estructura	`one = sym('3*x+2*y`
		`-z =10');`
		`two = sym('-x+3*y+2`
		`*z =5');`
		`three =sym('x - y`
		`- z = - 1');`
		`solve(one,two,three)`
		`ans =`
		`       x: [1x1 sym]`
		`       y: [1x1 sym]`
		`       z: [1x1 sym]`

`[A,B,C]= solve (S1,S2,S3)`	resuelve un sistema de ecuaciones y asigna las soluciones a nombres de variable definidos por el usuario. Los resultados se despliegan alfabéticamente	`one = sym('3*x+2*y`
		`-z =10');`
		`two = sym('-x+3*y+2`
		`*z =5');`
		`three = sym('x - y`
		`- z = -1');`
		`[x,y,z]=solve(one,`
		`two,three)`
		`x =`
		`-2`
		`y =`
		`5`
		`z =`
		`-6`

Ejercicio de práctica 11.4

Considere el siguiente sistema de ecuaciones lineales para usar en los problemas del 1 al 5:

$$5x + 6y - 3z = 10$$
$$3x - 3y + 2z = 14$$
$$2x - 4y - 12z = 24$$

1. Resuelva este sistema de ecuaciones mediante las técnicas del álgebra lineal discutidas en el capítulo 9.
2. Defina una ecuación simbólica que represente cada ecuación en el sistema de ecuaciones dado. Use la función **solve** para resolver para x, y y z.
3. Despliegue los resultados del problema 2 con la sintaxis del arreglo estructura.
4. Despliegue los resultados del problema 2 especificando los nombres de salida.
5. Agregue puntos decimales a los números en sus definiciones de ecuación y use **solve** de nuevo. ¿Cómo cambian sus respuestas?
6. Considere el siguiente sistema de ecuaciones no lineales:

$$x^2 + 5y - 3z^3 = 15$$
$$4x + y^2 - z = 10$$
$$x + y + z = 15$$

Resuelve el sistema no lineal con la función **solve**. Use la función **double** en sus resultados para simplificar la respuesta.

11.2.3 Sustitución

En particular para ingenieros o científicos, una vez que se tiene una expresión simbólica, con frecuencia se quiere sustituir valores en ella. Considere de nuevo la ecuación cuadrática:

```
E4 = sym('a*x^2+b*x+c')
```

Idea clave: si una variable no se menciona como variable simbólica en la ventana del área de trabajo, se debe encerrar en apóstrofes cuando se use en la función **subs**.

Existen algunas sustituciones que se pueden hacer. Por ejemplo, es posible que se quiera cambiar la variable **x** en la variable **y**. Para lograr esto, la función **subs** requiere tres entradas: la expresión a modificar, la variable a modificar y la nueva variable a insertar. Para sustituir **y** para todas las **x**, se usaría el comando

```
subs(E4,'x','y')
```

que regresa

```
ans =
a*(y)^2+b*(y)+c
```

La variable **E4** no se cambió; en vez de ello, la nueva información se almacenó en **ans**, o se le podría dar un nuevo nombre, como **E5**:

```
E5=subs(E4,'x','y')
E5 =
a*(y)^2+b*(y)+c
```

Al recordar **E4**, se ve que permanece invariable:

```
E4

E4 =
a*x^2+b*x+c
```

Para sustituir números, se usa el mismo procedimiento:

```
subs(E4,'x',3)
ans =
9*a+3*b+c
```

Como con otras operaciones simbólicas, si las variables se definieron anteriormente de manera explícita como simbólicas, no se requieren los apóstrofes. Por ejemplo,

```
syms a b c x
subs(E4,x,4)
```

regresa

```
ans =
16*a+4*b+c
```

Se pueden realizar múltiples sustituciones al mencionar las variables dentro de llaves, lo que define un arreglo celda:

```
subs(E4,{a,b,c,x},{1,2,3,4})

ans =
     27
```

Incluso se puede sustituir en arreglos numéricos. Por ejemplo, primero cree una nueva expresión que contenga sólo **x**:

```
E6=subs(E4,{a,b,c},{1,2,3})
```

Esto produce

```
E6 =
x^2+2*x+3
```

Ahora se define un arreglo de números y se les sustituye en **E6**:

```
numbers = 1:5;

subs(E6,x,numbers)
ans =
  6    11    18    27    38
```

No se podría realizar esto en un solo paso, porque cada uno de los elementos del arreglo celda almacenados entre llaves debe tener el mismo tamaño para que funcione la función **subs**.

Ejercicio de práctica 11.5

1. Con la función **subs**, sustituya **4** en cada expresión/ecuación definida en la práctica 11.1 para **x** (o **X**). Comente sus resultados.
2. Defina un vector **v** de los números pares del 0 al 10. Sustituya este vector en las cuatro versiones de expresión/ecuación: **ex1**, **EX1**, **eq1** y **EQ1**. ¿Esto funciona para las cuatro versiones de la expresión/ecuación? Comente sus resultados.

3. Sustituya los siguientes valores en las cuatro versiones de expresión/ecuación 4, **ex4**, **EX4**, **eq4** y **EQ4** (éste es un proceso de dos pasos porque **x** es un vector):

a = 3		**A = 3**
b = 4	o	**B = 4**
c = 5		**C = 5**
x = 1:0.5:5		**X = 1:0.5:5**

4. Verifique sus resultados para el problema 3 en la ventana del área de trabajo. ¿Qué tipo de variable es su resultado, doble o simbólico?

EJEMPLO 11.2

Uso de las matemáticas simbólicas para resolver un problema de balística

Se pueden usar las capacidades de las matemáticas simbólicas de MATLAB para explorar las ecuaciones que representan la ruta seguida por un proyectil sin combustible, como la bola de cañón que se muestra en la figura 11.4.

Se sabe por la física elemental que la distancia que recorre horizontalmente un proyectil es

$$dx = v_0 t \cos(\theta)$$

y la distancia recorrida verticalmente es

$$d_y = v_0 t \operatorname{sen}(\theta) - \frac{1}{2} g t^2$$

donde

v_0 = velocidad en el lanzamiento,
t = tiempo,
θ = ángulo de lanzamiento, y
g = aceleración debida a la gravedad.

Use estas ecuaciones y la capacidad simbólica de MATLAB para derivar una ecuación para la distancia que recorre horizontalmente el proyectil cuando golpea el suelo (el rango).

1. Establezca el problema.
   Encontrar la ecuación del rango.

Figura 11.4
El rango de un proyectil depende de la velocidad inicial y el ángulo de lanzamiento.

2. Describa las entradas y salidas.

   Entrada Ecuaciones para distancias horizontal y vertical

   Salida Ecuación para rango

3. Desarrolle un ejemplo a mano.

$$d_y = v_0 t \text{ sen}(\theta) - \frac{1}{2} g t^2 = 0$$

Reordene para obtener

$$v_0 t \text{ sen}(\theta) = \frac{1}{2} g t^2$$

Divida entre t y resuelva:

$$t = \frac{2 v_0 \text{ sen}(\theta)}{g}$$

Ahora sustituya esta expresión para t en la fórmula de la distancia horizontal para obtener

$$d_x = v_0 t \cos(\theta)$$

$$\text{rango} = v_0 * \left(\frac{2 v_0 \text{ sen}(\theta)}{g} \right) \cos(\theta)$$

Por la trigonometría se sabe que 2 sen θ cos θ es lo mismo que sen(2 θ), que permitiría una mayor simplificación si se desea.

4. Desarrolle una solución MATLAB.
   Defina primero las variables simbólicas:

```
syms v0 t theta g
```

A continuación defina la expresión simbólica para la distancia vertical recorrida:

```
Distancey = v0 * t *sin(theta) - 1/2*g*t^2;
```

Ahora defina la expresión simbólica para la distancia horizontal recorrida:

```
Distancex = v0 * t *cos(theta);
```

Resuelva la expresión de distancia vertical para el tiempo de impacto, pues la distancia vertical = 0 al impacto:

```
impact_time = solve(Distancey,t)
```

Esto regresa dos respuestas:

```
impact_time =
[          0]
[ 2*v0*sin(theta)/g]
```

Este resultado tiene sentido, pues la distancia vertical es cero al lanzamiento y de nuevo en el impacto. Sustituya el tiempo de impacto en la expresión de distancia horizontal. Dado que sólo se tiene interés en el segundo tiempo, necesitará usar **impact_time(2)**:

```
impact_distance = subs(Distancex,t,impact_time(2))
```

La sustitución resulta en una ecuación para la distancia que el proyectil recorre cuando golpea el suelo:

```
impact_distance =
2*v0^2*sin(theta)/g*cos(theta)
```

5. Ponga a prueba la solución.
Compare la solución MATLAB con la solución a mano. Ambos enfoques dan el mismo resultado.
MATLAB puede simplificar el resultado, aunque ya es bastante simple. Se elige usar el comando **simple** para demostrar todas las posibilidades. El comando

```
simple(impact_distance)
```

da los siguientes resultados:

```
simplify:          2*v0^2*sin(theta)/g*cos(theta)
radsimp:           2*v0^2*sin(theta)/g*cos(theta)
combine(trig):     v0^2*sin(2*theta)/g
factor:            2*v0^2*sin(theta)/g*cos(theta)
expand:            2*v0^2*sin(theta)/g*cos(theta)
combine:           v0^2*sin(2*theta)/g
convert(exp):      -i*v0^2*(exp(i*theta)-
                   1/exp(i*theta))/g*(1/2*exp(i*theta)+
                   1/2/exp(i*theta))
convert(sincos):   2*v0^2*sin(theta)/g*cos(theta)
convert(tan):      4*v0^2*tan(1/2*theta)/
                   (1+tan(1/2*theta)^2)^2/g*
                   (1-tan(1/2*theta)^2)
collect(v0):       2*v0^2*sin(theta)/g*cos(theta)
mwcos2sin:         2*v0^2*sin(theta)/g*cos(theta)
                   ans =
                   v0^2*sin(2*theta)/g
```

11.3 GRAFICACIÓN SIMBÓLICA

La caja de herramientas simbólica incluye un grupo de funciones que le permiten graficar funciones simbólicas. La más básica es **ezplot**.

11.3.1 La función ezplot

Considere una función simple de **x**, como

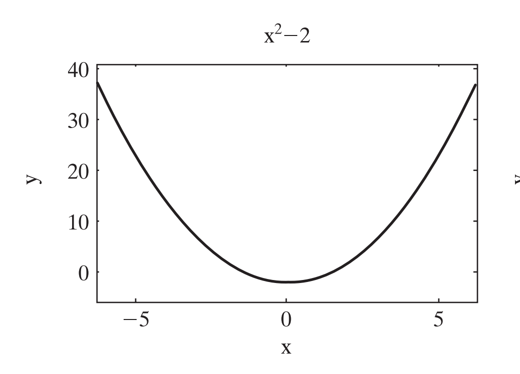
```
y=sym('x^2-2')
```

Para graficar esta función, use

```
ezplot(y)
```

La gráfica resultante se muestra en la figura 11.5. La función **ezplot** tiene por defecto una **x** que varía de -2π a $+2\pi$. MATLAB creó esta gráfica al elegir valores de **x** y calcular correspondientes valores de **y**, de modo que se produjo una curva suave. Note que la expresión graficada se despliega automáticamente como el título de una **ezplot**.

El usuario que no quiera aceptar los valores por defecto puede especificar los valores mínimo y máximo de **x** en el segundo campo de la función **ezplot**:



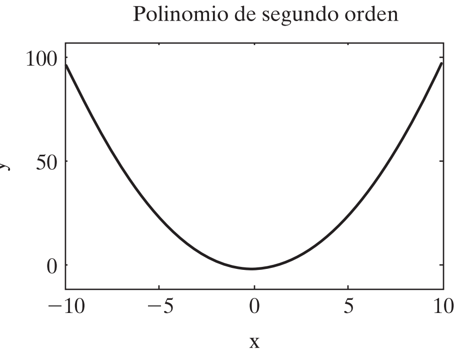


Figura 11.5
Las expresiones simbólicas se pueden graficar con **ezplot**. En la gráfica de la izquierda, el título por defecto es la expresión graficada y el rango por defecto es -2π a $+2\pi$. En la gráfica de la derecha, títulos, etiquetas y otras anotaciones se agregan a **ezplot** con las funciones MATLAB estándar.

```
ezplot(y,[-10,10])
```

Los valores se encierran en corchetes, lo que indica que son elementos en el arreglo que define los extremos de la gráfica. Además, puede especificar títulos, etiquetas de ejes y anotaciones, tal como hizo para otras gráficas MATLAB. Por ejemplo, para agregar un título y etiquetas a la gráfica, use

```
title('Polinomio de segundo orden')
xlabel('x')
ylabel('y')
```

La función **ezplot** también le permite graficar funciones implícitas de *x* y *y*, así como funciones paramétricas. Por ejemplo, considere la ecuación implícita

$$x^2 + y^2 = 1$$

que puede reconocer como la ecuación de un círculo de radio 1. Podría resolver para *y*, pero no es necesario con **ezplot**. Cualquiera de los comandos

```
ezplot('x^2 + y^2 =1',[-1.5,1.5])
ezplot('x^2 + y^2 -1',[-1.5,1.5])
```

y

```
z=sym('x^2 + y^2 -1')
ezplot(z,[-1.5,1.5])
```

se puede usar para crear la gráfica del círculo que se muestra en el lado izquierdo de la figura 11.6.

Otra forma de definir una ecuación es paramétricamente; esto es: definir ecuaciones separadas para *x* y para *y* en términos de una tercera variable. Un círculo se puede definir paramétricamente como

ecuaciones paramétricas: ecuaciones que definen x y y en términos de otra variable, usualmente t

$$x = \text{sen}(t)$$
$$y = \cos(t)$$

Para graficar el círculo paramétricamente con **ezplot**, mencione primero la expresión simbólica para *x* y luego para *y*:

```
ezplot('sen(x)','cos(x)')
```

Los resultados se muestran en el lado derecho de la figura 11.6.

Aunque la anotación de gráficas se realiza de la misma forma para gráficas simbólicas que para gráficas numéricas estándar, con la finalidad de graficar múltiples líneas en la misma gráfica, necesitará usar el comando **hold on**. Para ajustar colores, estilos de línea y estilos de marcador, use

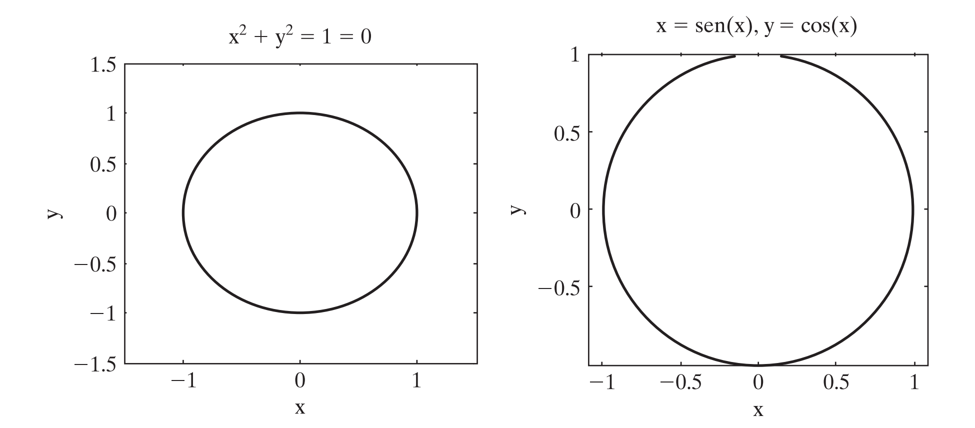


Figura 11.6
La función **ezplot** se puede usar para graficar funciones tanto implícitas como paramétricas, además de funciones de una sola variable.

las herramientas interactivas disponibles en la ventana de graficación. Por ejemplo, para graficar sen(x), sen($2x$) y sen($3x$) en la misma gráfica, primero defina algunas expresiones simbólicas:

```
y1=sym('sen(x)')
y2=sym('sen(2*x)')
y3=sym('sen(3*x)')
```

Luego grafique cada expresión:

```
ezplot(y1)
hold on
ezplot(y2)
ezplot(y3)
```

Los resultados se muestran en la figura 11.7. Para cambiar los colores de línea, los estilos de línea o los estilos de marcador, necesitará seleccionar la flecha en la barra de menú (en círculo en la figura) y luego seleccionar la línea que le gustaría editar. Una vez que seleccione la línea, haga clic derecho para activar el menú de edición. No olvide emitir el comando

```
hold off
```

una vez que realice la graficación.

Sugerencia

La mayoría de las funciones simbólicas le permitirán ingresar una variable simbólica que represente una función o ingresar la función misma encerrada entre apóstrofes. Por ejemplo,

```
y=sym('x^2-1')
ezplot(y)
```

es equivalente a

```
ezplot('x^2-1')
```

Ejercicio de práctica 11.6

Asegúrese de agregar títulos y etiquetas de eje a todas sus gráficas.

1. Use **ezplot** para graficar **ex1** desde -2π hasta $+2\pi$.

2. Use **ezplot** para graficar **EX1** desde -2π hasta $+2\pi$.
3. Use **ezplot** para graficar **ex2** desde -10 hasta $+10$.
4. Use **ezplot** para graficar **EX2** desde -10 hasta $+10$.
5. ¿Por qué no se pueden graficar las ecuaciones con sólo una variable?
6. Use **ezplot** para graficar **ex6** desde -2π hasta $+2\pi$.
7. Use **ezplot** para graficar cos(x) desde -2π hasta $+2\pi$. No defina una expresión para cos(x); sólo ingrésela en **ezplot** como una cadena carácter:

   ezplot('cos(x)')
8. Use **ezplot** para crear una gráfica implícita de **x^2 – y^4=5**.
9. Use **ezplot** para graficar sen(x) y cos(x) en la misma gráfica. Use las herramientas interactivas de graficación para cambiar el color de la gráfica seno.
10. Use **ezplot** para crear una gráfica paramétrica de x = sen(t) y y = 3 cos(t).

11.3.2 Gráficas simbólicas adicionales

Las funciones de graficación simbólica adicionales que reflejan las funciones que se usan en las opciones de graficación numéricas MATLAB se citan en la tabla 11.3.

Para demostrar cómo trabajan las funciones de graficación de superficie tridimensional (**ezmesh**, **ezmesch**, **ezsurf** y **ezsurfc**), primero defina una versión simbólica de la función **peaks**:

```
z1 =sym('3*(1-x)^2*exp(-(x^2) - (y+1)^2)')
z2=sym('- 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2)')
z3=sym('- 1/3*exp(-(x+1)^2 - y^2)')
z=z1+z2+z3
```

Esta función se descompone en tres partes para hacerla más fácil de ingresar en la computadora. Note que en estas expresiones no se usan operadores "punto", pues todos son simbólicos. Las funciones **ezplot** funcionan de manera similar a sus contrapartes numéricas:

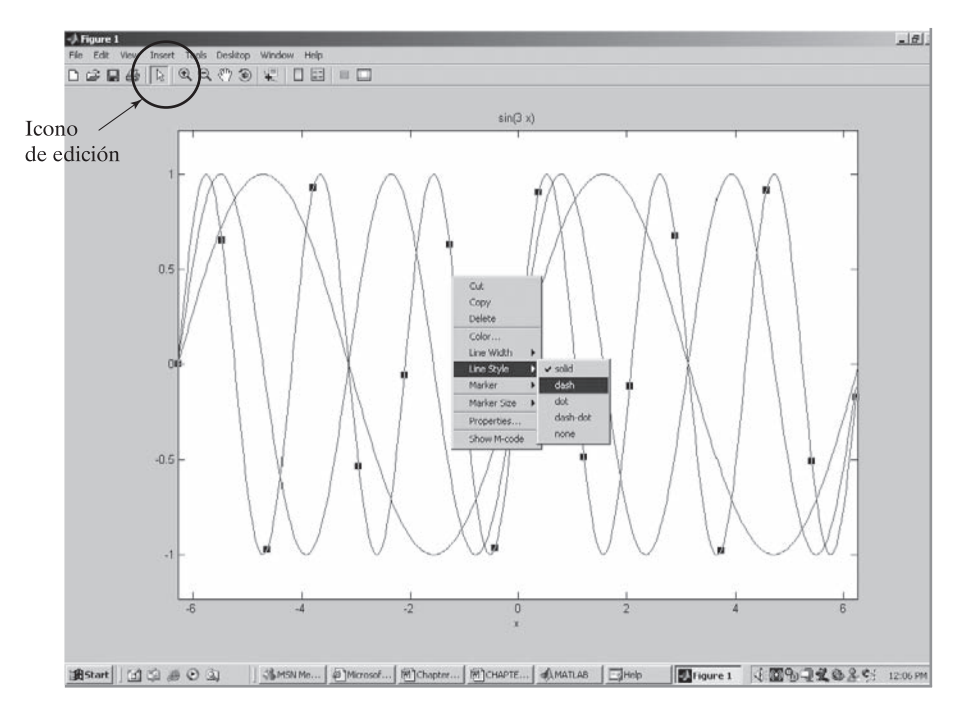


Figura 11.7
Use las herramientas interactivas de graficación para ajustar estilo de línea, color y marcadores.

Idea clave: la mayoría de las funciones de graficación MATLAB para arreglos tienen funciones correspondientes para aplicaciones simbólicas.

Tabla 11.3 Funciones de graficación simbólica

ezplot	graficador de función	si *z* es una función de *x*: **ezplot(z)**
ezmesh	gráfica malla	si *z* es una función de *x* y *y*: **ezmesh(z)**
ezmeshc	graficador combinado malla y contorno	si *z* es una función de *x* y *y*: **ezmeshc(z)**
ezsurf	graficador de superficie	si *z* es una función de *x* y *y*: **ezsurf(z)**
ezsurfc	graficador combinado superficie y contorno	si *z* es una función de *x* y *y*: **ezsurfc(z)**
ezcontour	graficador contorno	si *z* es una función de *x* y *y*: **ezcontour(z)**
ezcontourf	graficador contorno lleno	si *z* es una función de *x* y *y*: **ezcontourf(z)**
ezplot3	graficador curva paramétrica tridimensional	si *x* es una función de *t*, si *y* es una función de *t* y si *z* es una función de *t*: **ezplot3(x,y,z)**
ezpolar	graficador coordenada polar	si *r* es una función de θ: **ezpolar(r)**

```
subplot(2,2,1)
ezmesh(z)
title('ezmesh')
subplot(2,2,2)
ezmeshc(z)
title('ezmeshc')
subplot(2,2,3)
ezsurf(z)
title('ezsurf')
subplot(2,2,4)
ezsurfc(z)
title('ezsurfc')
```

Las gráficas que resultan de estos comandos se muestran en la figura 11.8. Cuando se crearon las mismas gráficas mediante un enfoque MATLAB estándar, fue necesario definir un arreglo de valores *x* y *y*, ponerlos en malla y calcular los valores de *z* sobre la base de los arreglos bidimensionales. La capacidad de graficación simbólica contenida en la caja de herramientas simbólica hace mucho más sencilla la creación de estas gráficas.

Todas estas gráficas se pueden anotar con el uso de las funciones MATLAB estándar, como **title**, **xlabel**, **text**, etcétera.

Las gráficas bidimensionales y de contorno también son similares a sus contrapartes numéricas:

```
subplot(2,2,1)
ezcontour(z)
title ('ezcontour')
subplot(2,2,2)
ezcontourf(z)
title('ezcontourf')
subplot(2,2,3)
z=sym('sin(x)')
ezpolar(z)
title('ezpolar')
subplot(2,2,4)
ezplot(z)
title('ezplot')
```

Estas gráficas de contorno son una representación bidimensional de la función **peaks** tridimensional y se muestran en la figura 11.9. La gráfica polar requiere definir una nueva función, que también se graficó con el uso de la **ezplot** básica.

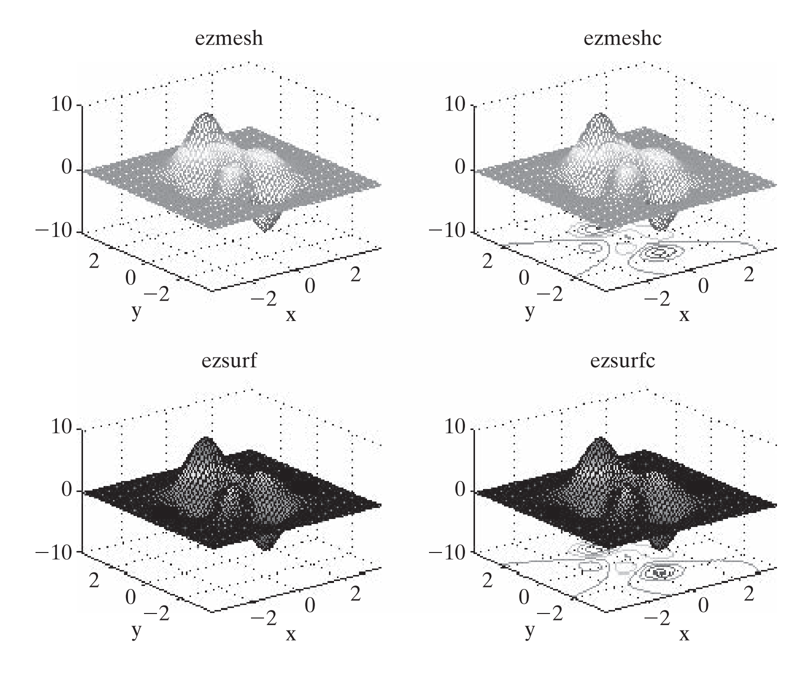


Figura 11.8
Ejemplos de gráficas de superficie simbólica tridimensional.

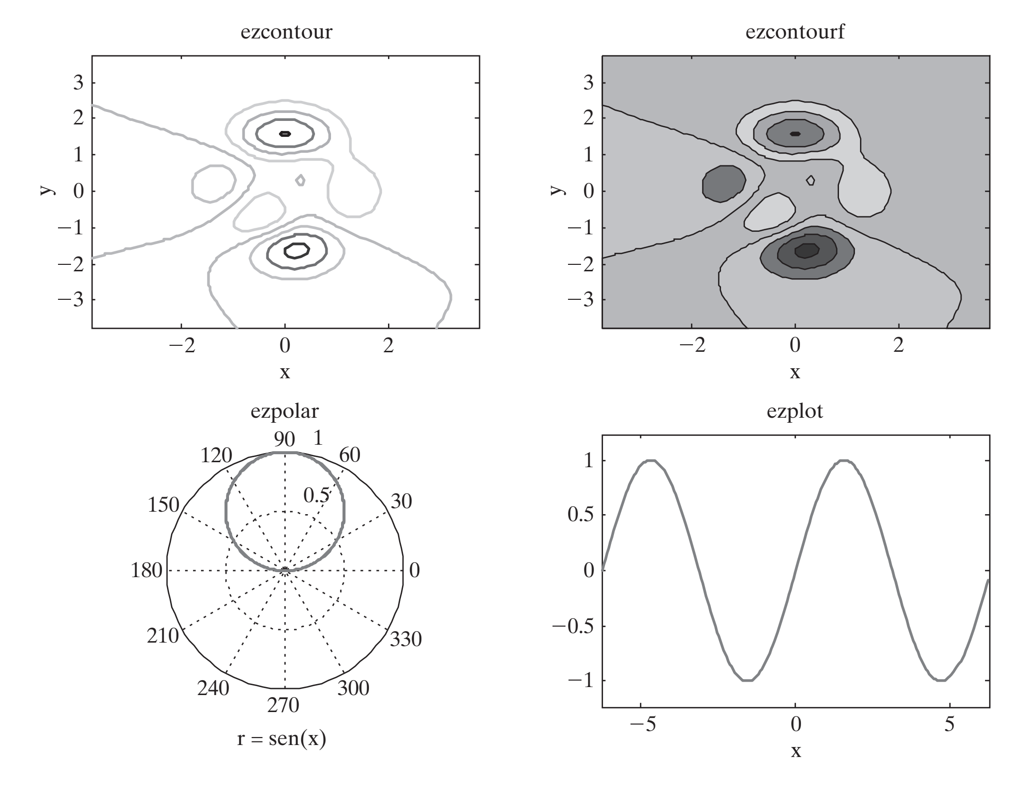


Figura 11.9
Ejemplos de gráficas simbólicas bidimensional y de contorno.

Ejercicio de práctica 11.7

Cree una expresión simbólica para $Z = \text{sen}\left(\sqrt{X^2 + Y^2}\right)$.

1. Use **ezmesh** para crear una gráfica de malla de **Z**. Asegúrese de agregar un título y etiquetas de eje.
2. Use **ezmeshc** para crear una combinación de gráfica de malla y contorno de **Z**. Asegúrese de agregar un título y etiquetas de eje.
3. Use **ezsurf** para crear una gráfica de superficie de **Z**. Asegúrese de agregar un título y etiquetas de eje.
4. Use **ezsurfc** para crear una combinación de gráfica de superficie y de contorno de **Z**. Asegúrese de agregar un título y etiquetas de eje.
5. Use **ezcontour** para crear una gráfica de contorno de **Z**. Asegúrese de agregar un título y etiquetas de eje.
6. Use **ezcontourf** para crear una gráfica de contorno llena de **Z**. Asegúrese de agregar un título y etiquetas de eje.
7. Use **ezpolar** para crear una gráfica polar de x sen(x). No defina una expresión simbólica, ingrese esta expresión directamente en **ezpolar**:

   ezpolar('x*sen(x)')

   Asegúrese de agregar un título.
8. La función **ezplot3** requiere definir tres variables como función de una cuarta. Para hacerlo, primero defina t como una variable simbólica, y luego haga

$$x = t$$
$$y = \text{sen}(t)$$
$$z = \cos(t)$$

   Use **ezplot3** para graficar esta función paramétrica desde 0 hasta 30. Es posible que tenga problemas al crear gráficas **ezplot3** dentro de ventanas de subgráfica, debido a una idiosincrasia de programa MATLAB. Versiones posteriores pueden corregir este problema.

EJEMPLO 11.3

Uso de graficación simbólica para ilustrar un problema de balística

En el ejemplo 11.2 se usaron las capacidades simbólicas de MATLAB para derivar una ecuación para la distancia que recorre un proyectil antes de golpear el suelo. La fórmula de distancia horizontal

$$dx = v_0 t \cos(\theta)$$

y la fórmula de distancia vertical

$$d_y = v_0 t \,\text{sen}(\theta) - \frac{1}{2} g t^2$$

donde

v_0	=	velocidad en el lanzamiento,
t	=	tiempo,
θ	=	ángulo de lanzamiento, y
g	=	aceleración debida a la gravedad.

se combinaron para dar

$$\text{rango} = v_0\left(\frac{2v_0 \operatorname{sen}(\theta)}{g}\right)\cos(\theta)$$

Con las capacidades de graficación simbólica de MATLAB, cree una gráfica que muestre el rango recorrido para ángulos desde 0 hasta $\pi/2$. Suponga una velocidad inicial de 100 m/s y una aceleración debida a la gravedad de 9.8 m/s^2.

1. Establezca el problema.
   Grafique el rango como función del ángulo de lanzamiento.
2. Describa las entradas y salidas.

   Entrada Ecuación simbólica para rango

   $$v_0 = 100 \text{ m/s}$$
   $$g = 9.8 \text{ m/s}^2$$

   Salida Gráfica de rango contra ángulo
3. Desarrolle un ejemplo a mano.

   $$\text{rango} = v_0\left(\frac{2v_0 \operatorname{sen}(\theta)}{g}\right)\cos(\theta)$$

   Por la trigonometría se sabe que 2 sen θ cos θ es igual a sen(2θ). Por tanto, se puede simplificar el resultado a

   $$\text{rango} = \frac{v_0^2}{g}\operatorname{sen}(2\theta)$$

   Con esta ecuación, es fácil calcular unos cuantos puntos de datos:

Ángulo	**Rango, m**
0	0
$\pi/6$	884
$\pi/4$	1020
$\pi/3$	884
$\pi/2$	0

   El rango parece aumentar con ángulo creciente y luego disminuye de nuevo a cero cuando el cañón apunta directo hacia arriba.
4. Desarrolle una solución MATLAB.
   Primero se necesita modificar la ecuación del ejemplo 11.2 para incluir la velocidad de lanzamiento y la aceleración debida a la gravedad. Recuerde que

   ```
   impact_distance =
   2*v0^2*sin(theta)/g*cos(theta)
   ```

   Use la función **subs** para sustituir los valores numéricos en la ecuación:

   ```
   impact_100 = subs(impact_distance,{v0,g},{100, 9.8})
   ```

Esto regresa

```
impact_100 =
100000/49*sin(theta)*cos(theta)
```

Finalmente, grafique los resultados y agregue un título y etiquetas:

```
ezplot(impact_100,[0, pi/2])
title('Distancia máxima recorrida por el proyectil')
xlabel('ángulo, radianes')
ylabel('rango, m')
```

Esto genera la figura 11.10.

5. Ponga a prueba la solución.
   La solución MATLAB concuerda con la solución a mano. El rango es cero cuando el cañón apunta recto hacia arriba y también es cero cuando se apunta horizontalmente. El rango parece tener un pico a un ángulo de aproximadamente 0.8 radianes, que corresponde más o menos a 45 grados.

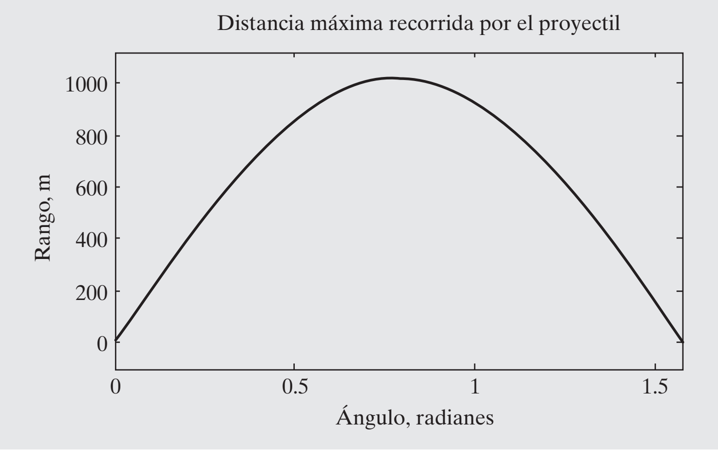


Figura 11.10
Rango de proyectil.

11.4 CÁLCULO

La caja de herramientas simbólica de MATLAB permite al usuario diferenciar simbólicamente y realizar integraciones. Esto hace posible encontrar soluciones analíticas, en lugar de aproximaciones numéricas, para muchos problemas.

11.4.1 Diferenciación

El cálculo diferencial se estudia extensamente en el cálculo de primer semestre. La derivada se puede considerar como la pendiente de una función o como la tasa de cambio de la función. Por ejemplo, considere un auto de carreras. La velocidad del automóvil se puede aproximar como el cambio en distancia dividido por el cambio en tiempo. Suponga que, durante una carrera, el automóvil parte lentamente y alcanza su mayor rapidez en la línea final. Desde luego, para evitar precipitarse sobre las gradas, el automóvil debe frenar entonces hasta detenerse final-

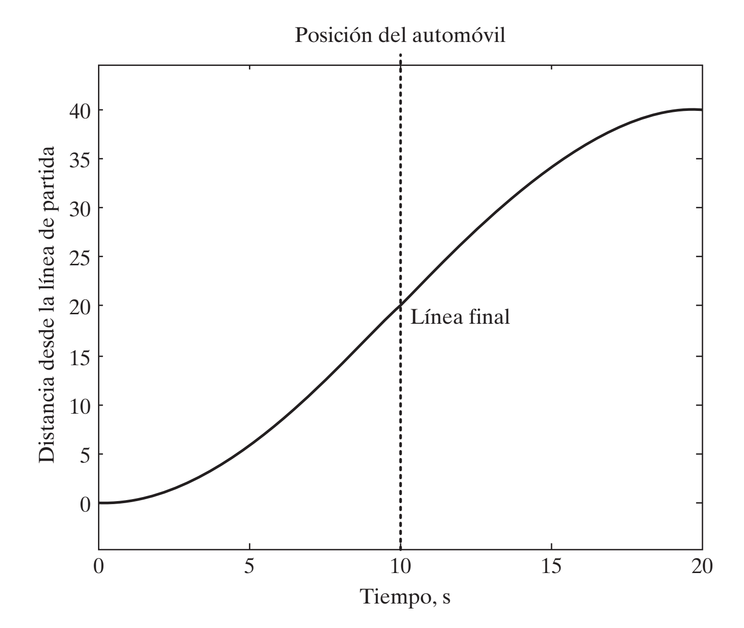


Figura 11.11
Posición de un auto de carreras. El automóvil acelera hasta que alcanza la línea final. Luego frena hasta detenerse. (La línea punteada que indica la línea final se agregó después de creada la gráfica.)

mente. Se debe modelar la posición del automóvil con una onda seno, como se muestra en la figura 11.11. La ecuación relevante es

$$d = 20 + 20\,\text{sen}\left(\frac{\pi(t - 10)}{20}\right)$$

La gráfica en la figura 11.11 se creó con **ezplot** y matemáticas simbólicas. Primero se define una expresión simbólica para distancia:

```
dist = sym('20+20*sin(pi*(t-10)/20)')
```

Una vez que se tiene la expresión simbólica, se le puede sustituir en la función **ezplot** y anotar la gráfica resultante:

```
ezplot(dist,[0,20])
title('Posición del auto')
xlabel('tiempo, s')
ylabel('Distancia desde la línea de partida')
text(10,20,'Línea final')
```

MATLAB incluye una función llamada **diff** para encontrar la derivada de una expresión simbólica. (La palabra *diferencial* es otro término para la derivada.) La velocidad es la derivada de la posición, de modo que, para encontrar la ecuación de velocidad del automóvil, se usará la función **diff**:

```
velocity=diff(dist)
velocity =
cos(1/20*pi*(t-10))*pi
```

Se puede usar la función **ezplot** para graficar la velocidad:

```
ezplot(velocity,[0,20])
title('Velocidad del auto de carreras')
xlabel('tiempo, s')
```

```
ylabel('velocidad, distancia/tiempo')
text(10,3,'Línea final')
```

Los resultados se muestran en la figura 11.12.

La aceleración del auto de carreras es el cambio en la velocidad dividida por el cambio en tiempo, de modo que la aceleración es la derivada de la función velocity:

```
acceleration=diff(velocity)
acceleration =
-1/20*sin(1/20*pi*(t-10))*pi^2
```

La gráfica de la aceleración (figura 11.13) también se creó con el uso de la función graficación simbólica:

```
ezplot(acceleration,[0,20])
title(' Aceleración del auto de carreras')
xlabel('tiempo, s')
ylabel('aceleración, velocidad/tiempo')
text(10,0,'Línea final')
```

derivada: la tasa instantánea de cambio de una variable con respecto a una segunda variable

La aceleración es la primera derivada de la velocidad y la segunda derivada de la posición. MATLAB ofrece muchas formas ligeramente diferentes para encontrar tanto primeras derivadas como *n*-ésimas derivadas. (Véase la tabla 11.4.)

Si se tiene una ecuación más complicada con múltiples variables, como

```
y=sym('x^2+t-3*z^3')
```

MATLAB calculará la derivada con respecto a **x**, la variable por defecto:

```
diff(y)
ans =
2*x
```

El resultado es la tasa de cambio de **y** conforme **x** cambia (si se mantienen constantes todas las otras variables). Usualmente esto se expresa como $\partial y/\partial x$ y se llama *derivada parcial*.

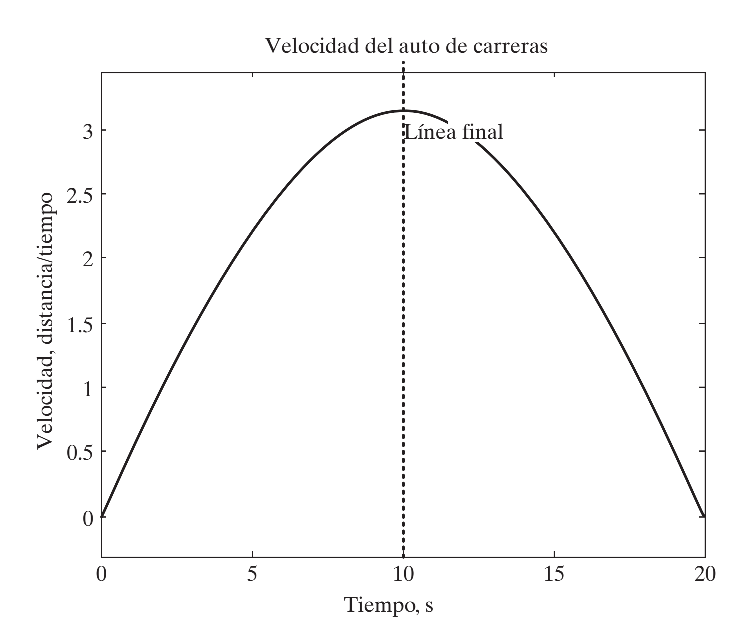


Figura 11.12
La velocidad máxima se alcanza en la línea final.

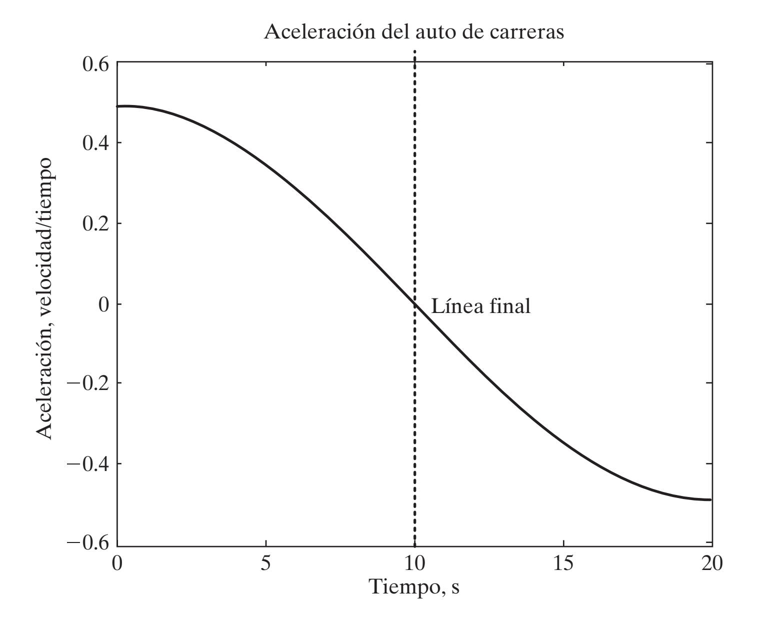


Figura 11.13
El auto de carreras acelera hasta la línea final y luego desacelera. La aceleración en la línea final es cero.

Si se quiere ver cómo cambia **y** con respecto a otra variable, como **t**, se debe especificar en la función **diff** (recuerde que si **t** se definió previamente como una variable simbólica, no es necesario encerrarla entre apóstrofes):

```
diff(y,'t')
ans =
1
```

De igual modo, para ver cómo cambia **y** con **z** cuando todo lo demás se mantiene constante, se usa

```
diff(y,'z')
ans =
-9*z^2
```

Tabla 11.4 Diferenciación simbólica

`diff(f)`	regresa la derivada de la expresión **f** con respecto a la variable independiente por defecto	`y=sym('x^3+z^2')`
		`diff(y)`
		`ans =`
		`3*x^2`

`diff(f,'t')`	regresa la derivada de la expresión **f** con respecto a la variable **t**	`y=sym('x^3+z^2')`
		`diff(y,'z')`
		`ans =`
		`2*z`

`diff(f,n)`	regresa la *n*-ésima derivada de la expresión **f** con respecto a la variable independiente por defecto	`y=sym('x^3+z^2')`
		`diff(y,2)`
		`ans =`
		`6*x`

`diff(f,'t',n)`	regresa la *n*-ésima derivada de la expresión **f** con respecto a la variable **t**	`y=sym('x^3+z^2')`
		`diff(y,'z',2)`
		`ans =`
		`2`

Para encontrar derivada de orden superior, se puede anidar la función **diff** o especificar el orden de la derivada en la función **diff**. Cualquiera de los enunciados

Idea clave: la integración es lo opuesto de sacar la derivada.

```
diff(y,2)
```

y

```
diff(diff(y))
```

regresa el mismo resultado:

```
ans =
2
```

Note que, aunque el resultado parezca ser un número, es una variable simbólica. Con la finalidad de usarla en un cálculo MATLAB, necesitará convertirla a un número punto flotante de precisión doble.

Si quiere obtener una derivada superior de **y** con respecto a una variable que no sea por defecto, necesita especificar tanto el grado de la derivada como la variable. Por ejemplo, para encontrar la segunda derivada de **y** con respecto a **z**, escriba

```
diff(y,'z',2)
ans =
-18*z
```

Ejercicio de práctica 11.8

1. Encuentre la primera derivada con respecto a x de las siguientes expresiones:

   $x^2 + x + 1$
   $\text{sen}(x)$
   $\tan(x)$
   $\ln(x)$

2. Encuentre la primera derivada parcial con respecto a x de las siguientes expresiones:

   $ax \quad + bx + c$
   $x^{0.5} - 3y$
   $\tan(x + y)$
   $3x + 4y - 3xy$

3. Encuentre la segunda derivada con respecto a x para cada una de las expresiones del problema 1 y 2.
4. Encuentre la primera derivada con respecto a y para las siguientes expresiones:

   $y \quad - 1$
   $2y + 3x^2$
   $ay + bx + cz$

5. Encuentre la segunda derivada con respecto a y para cada una de las expresiones del problema 4.

EJEMPLO 11.4

Uso de matemáticas simbólicas para encontrar el ángulo de lanzamiento óptimo

En el ejemplo 11.3, se usó la capacidad de graficación simbólica de MATLAB para crear una gráfica de rango contra ángulo de lanzamiento, con base en la fórmula de rango derivada en el ejemplo 11.2, a saber,

$$\text{rango} = v_0\left(\frac{v_0\,\text{sen}(\theta)}{g}\right)\cos(\theta)$$

donde

v_0 = velocidad en el lanzamiento, la cual elegimos sea 100 m/s,
θ = ángulo de lanzamiento, y
g = aceleración debida a la gravedad.

Use la capacidad simbólica de MATLAB para encontrar el ángulo al que ocurre el rango máximo y encontrar el rango máximo.

1. Establezca el problema.
   Encontrar el ángulo al que ocurre el rango máximo.
   Encontrar el rango máximo.

2. Describa las entradas y salidas.

   Entrada Ecuación simbólica para rango

   $v_0 = 100$ m/s

   $g = 9.8$ m/s^2

   Salida El ángulo al que ocurre el rango máximo
   El rango máximo

3. Desarrolle un ejemplo a mano.
   A partir de la gráfica de la figura 11.14, el rango máximo parece ocurrir a un ángulo de lanzamiento de aproximadamente 0.7 o 0.8 radianes, y la altura máxima parece ser aproximadamente 1000 m.

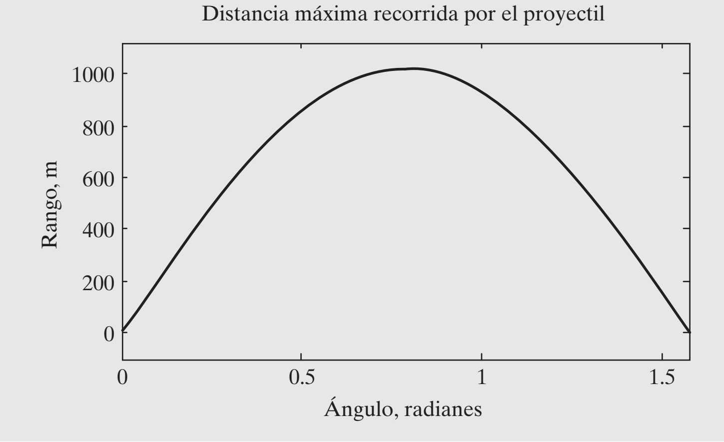


Figura 11.14
El rango del proyectil como una función del ángulo de lanzamiento.

4. Desarrolle un ejemplo a mano.
   Recuerde que la expresión simbólica para la distancia de impacto con v_0 y g definida como 100 m/s y 9.8 m/s^2, respectivamente, es

```
impact_100 =
100000/49*sin(theta)*cos(theta)
```

   A partir de la gráfica, se puede ver que la distancia máxima ocurre cuando la pendiente es igual a cero. La pendiente es la derivada de **impact_100**, así que necesita igualar la derivada a cero y resolver. Dado que MATLAB supone automáticamente que una expresión es igual a cero, se tiene

```
max_angle=solve(diff(impact_100))
```

   que regresa el ángulo al que ocurre la altura máxima:

```
max_angle =
[ -1/4*pi]
[  1/4*pi]
```

   Existen dos resultados, pero sólo tiene interés el segundo, que se puede sustituir en la expresión para el rango:

```
max_distance = subs(impact_100,theta,max_angle(2))
```

11.4.2 Integración

La integración se puede considerar como lo opuesto de la diferenciación (encontrar una derivada) e incluso a veces se le llama antiderivada. Comúnmente se le visualiza como el área bajo una curva. Por ejemplo, el trabajo realizado por un pistón conforme se mueve arriba y abajo se puede calcular al obtener la integral de P con respecto a V; esto es,

$$W = \int P\,dV$$

Para efectuar el cálculo, es necesario saber cómo cambia P con V. Si, por ejemplo, P es una constante, se podría crear la gráfica que se muestra en la figura 11.15.

El trabajo consumido o producido conforme se mueve el pistón es el área bajo la curva desde el volumen inicial hasta el volumen final. Por ejemplo, si se mueve el pistón desde 1 cm^3 hasta 4 cm^3, el trabajo correspondería al área que se muestra en la figura 11.16.

Como puede saber a partir de un curso de cálculo integral (usualmente Cálculo II), la integración es bastante simple:

$$W = \int_1^4 P\,dV = P\int_1^4 dV = PV\big|_1^4 = PV_4 - PV_1 = P\,\Delta V$$

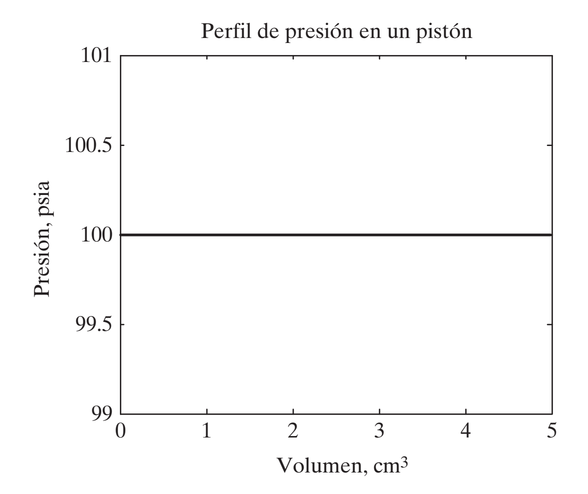
Si

$$P = 100 \text{ psia},$$

entonces

$$W = 3 \text{ cm}^3 \times 100 \text{ psia}$$

La caja de herramientas simbólica le permite obtener fácilmente integrales de algunas funciones muy complicadas. Por ejemplo, si quiere encontrar una integral indefinida (una integral para la que no se especifican los valores frontera de la variable), puede usar la función **int**. Primero se necesita especificar una función:



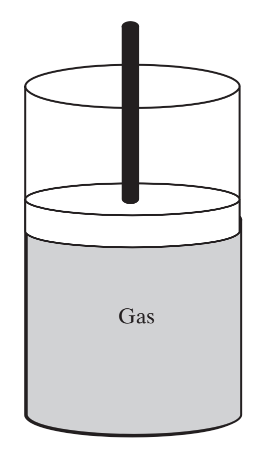


Figura 11.15
El perfil de presión en un pistón. En este ejemplo, la presión es constante.

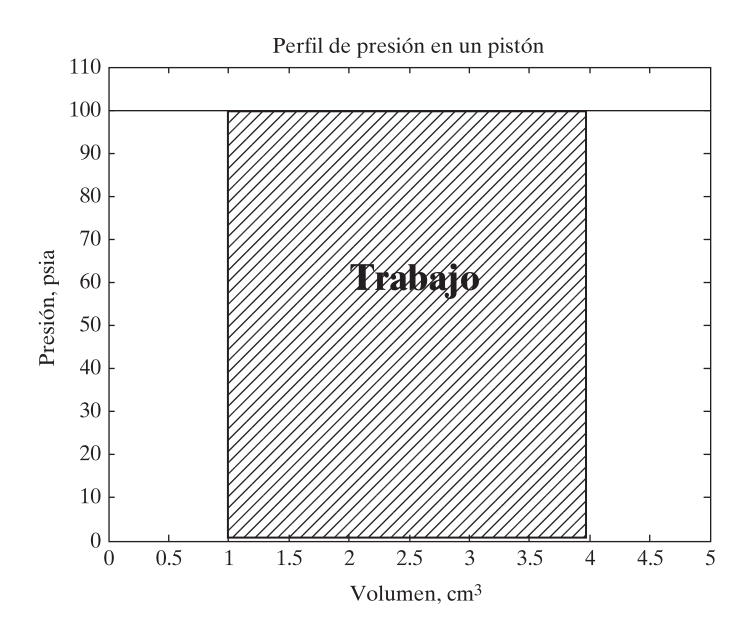


Figura 11.16
El trabajo producido en un pistón es el área bajo la curva.

```
y=sym('x^3 + sen(x)')
```

Para encontrar la integral indefinida, escriba

```
int(y)
ans =
1/4*x^4-cos(x)
```

La función **int** usa **x** como la variable por defecto. Por ejemplo, si se define una función con dos variables, la función **int** encontrará la integral con respecto a **x** o la variable más cercana a **x**:

```
y=sym('x^3 +sen(t)')
int(y)
ans =
1/4*x^4+sen(t)*x
```

Si se quiere obtener la integral con respecto a una variable definida por el usuario, dicha variable necesita especificarse en el segundo campo de la función **int**:

```
int(y,'t')
ans =
x^3*t-cos(t)
```

Para encontrar la integral definida es necesario especificar el rango de interés. Considere esta expresión:

```
y =sym('x^2')
```

Si no se especifica el rango de interés, se obtiene

```
int(y)
ans =
1/3*x^3
```

Se podría evaluar esto desde 2 hasta 3 al usar la función **subs**:

```
yy=int(y)
yy =
1/3*x^3
subs(yy,3)-subs(yy,2)
ans =
  6.3333
```

Un enfoque más simple es especificar las fronteras en la función **int**:

```
int(y,2,3)
ans =
19/3
double(ans)
ans =
  6.3333
```

Si se quiere especificar tanto la variable como las fronteras, es necesario mencionarlas:

```
y=sym('sen(x)+cos(z)')
int(y,'z',2,3)
ans =
sen(x)+sen(3)-sen(2)
```

Las fronteras pueden ser numéricas o pueden ser variables simbólicas:

```
int(y,'z','b','c')
ans =
sen(x)*c+sen(c)-sen(x)*b-sen(b)
```

La tabla 11.5 menciona las funciones MATLAB que tienen relación con la integración.

Tabla 11.5 Integración simbólica

`int(f)`	regresa la integral de la expresión **f** con respecto a la variable independiente por defecto	`y=sym('x^3+z^2')`
		`int(y)`
		`ans =`
		`1/4*x^4+z^2*x`

`int(f,'t')`	regresa la integral de la expresión **f** con respeto a la variable **t**	`y=sym('x^3+z^2')`
		`int(y,'z')`
		`ans =`
		`x^3*z+1/3*z^3`

`int(f,a,b)`	regresa la integral, con respecto a la variable por defecto, de la expresión **f** entre las fronteras numéricas **a** y **b**	`y=sym('x^3+z^2')`
		`int(y,2,3)`
		`ans =`
		`65/4+z^2`

`int(f,'t',a,b)`	regresa la integral, con respecto a la variable **t**, de la expresión **f** entre las fronteras numéricas **a** y **b**	`y=sym('x^3+z^2')`
		`int(y,'z',2,3)`
		`ans =`
		`x^3+19/3`

`int(f,'t',a,b)`	regresa la integral, con respecto a la variable **t**, de la expresión **f** entre las fronteras simbólicas **a** y **b**	`y=sym('x^3+z^2')`
		`int(y,'z','a','b')`
		`ans =`
		`x^3*(b-a)+1/3*b^3-`
		`1/3*a^3`

Ejercicio de práctica 11.9

1. Integre las siguientes expresiones con respecto a x:

   $x^2 + x + 1$

   $\text{sen}(x)$

   $\tan(x)$

   $\ln(x)$

2. Integre las siguientes expresiones con respeto a x:

   $ax^2 + bx + c$

   $x^{0.5} - 3y$

   $\tan(x + y)$

   $3x + 4y - 3xy$

3. Realice una integración doble con respecto a x para cada una de las expresiones de los problemas 1 y 2.
4. Integre las siguientes expresiones con respecto a y:

   $y^2 - 1$

   $2y + 3x^2$

   $ay + bx + cz$

5. Realice una integración doble con respecto a y para cada una de las expresiones en el problema 4.
6. Integre cada una de las expresiones en el problema 1 con respecto a x desde 0 hasta 5.

EJEMPLO 11.5

Uso de matemáticas simbólicas para encontrar el trabajo producido en un pistón

Los pistones se usan en un amplio rango de instrumentación científica y dispositivos de ingeniería. Probablemente el más presente es el motor de combustión interna (figura 11.17), que por lo general, usa cuatro para ocho cilindros.

El trabajo producido por un pistón depende de la presión dentro del cilindro y de la cantidad que se mueve el pistón, lo que resulta en un cambio en el volumen dentro del cilindro. Matemáticamente,

$$W = \int P\, dV$$

Para integrar esta ecuación, necesita entender cómo cambia la presión con el volumen. Se puede modelar la mayoría de los gases de combustión, como el aire, y suponer que siguen la ley del gas ideal

$$PV = nRT$$

donde

P = presión, kPa,
V = volumen, m^3,
n = número de moles, kmol,
R = constante universal de los gases, 8.314 kPa m^3/kmol K, y
T = temperatura, K.

Si supone que hay 1 mol de gas a 300 K y que la temperatura permanece constante durante el proceso, se pueden usar estas ecuaciones para calcular el trabajo realizado sobre el gas o producido por el gas conforme se expande o contrae entre dos volúmenes conocidos.

1. Establezca el problema.
   Calcular el trabajo realizado por mol en un pistón isotérmico (temperatura constante) conforme el gas se expande o contrae entre dos volúmenes conocidos.
2. Describa las entradas y salidas.

   Entrada

   Temperatura = 300 K

   Constante universal de los gases = 8.314 kPa m^3/kmol K = 8.314 kJ/kmol K

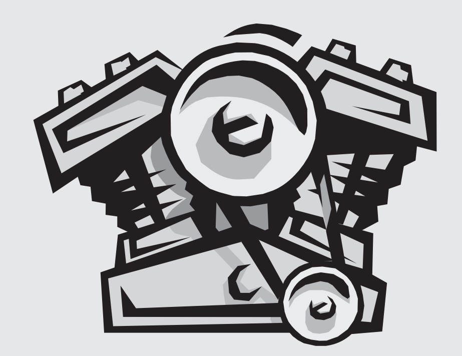


Figura 11.17
Motor de combustión interna.

Valores arbitrarios de volumen inicial y final; para este ejemplo, se usarán

$$\text{volumen inicial} = 1 \text{ m}^3$$

$$\text{volumen final} = 5 \text{ m}^3$$

Salida
Trabajo producido por el pistón, en kJ

3. Desarrolle un ejemplo a mano.
   Primero necesitará resolver la ley del gas ideal para P:

$$PV = nRT$$
$$P = nRT/V$$

Dado que n, R y T son constantes durante el proceso, ahora se puede realizar la integración:

$$W = \int \frac{nRT}{V} dV = nRT \int \frac{dV}{V} = nRT \ln\left(\frac{V_2}{V_1}\right)$$

Al sustituir los valores, se encuentra que

$$W = 1 \text{ kmol} \times 8.314 \text{ kJ/kmol K} \times 300 \text{ K} \times \ln\left(\frac{V_2}{V_1}\right)$$

Si se usan los valores arbitrarios $V_1 = 1 \text{ m}^3$ y $V_2 = 5 \text{ m}^3$, entonces el trabajo se convierte en

$$W = 4014 \text{ kJ}$$

Puesto que el trabajo es positivo, se produce *por* (no *sobre*) el sistema.

4. Desarrolle una solución MATLAB.
   Primero necesitará resolver la ley del gas ideal para presión. El código

```
syms P V n R T V1 V2             %Defina variables
ideal_gas_law=sym('P*V=n*R*T')  %Defina ley del gas ideal
P=solve(ideal_gas_law,'P')      %Resuelva para P
```

regresa

```
P =
n*R*T/V
```

Una vez que se tiene la ecuación para P, se puede integrar. El comando

```
W=int(P,V,V1,V2)                 %Integra P con respecto
                                 %a V desde V1 hasta V2
```

regresa

```
W =
n*R*T*log(V2)-n*R*T*log(V1)
```

Finalmente, se pueden sustituir los valores en la ecuación. Escriba

```
work=subs(W,{n,R,V1,V2,T},{1,8.314,1,5,300.0})
```

para obtener

```
work =
  4.0143e+003
```

5. Ponga a prueba la solución.
La prueba más obvia es comparar las soluciones a mano y computadora. Sin embargo, la misma respuesta con ambas técnicas sólo significa que se hicieron los cálculos de la misma manera. Una forma de verificar la razonabilidad sería crear una gráfica *PV* y estimar el área bajo la curva.

Para crear la gráfica, necesitará regresar a la ecuación para *P* y sustituir valores para *n*, *R* y *T*:

```
p=subs(P,{n,R,T},{1,8.314, 300})
```

Esto regresa la siguiente ecuación para *P*:

```
p =
12471/5/V
```

Ahora se puede usar **ezplot** para crear una gráfica de *P* contra *V* (véase la figura 11.18):

```
ezplot(p,[1,5])  %Grafica la presión contra V
title('La presión cambia con volumen para un sistema isotérmico')
xlabel('Volumen')
ylabel('Presión, psia')
xlabel('Volumen, cm^3')
axis([1,5,0,2500])
```

Para estimar el trabajo se podría encontrar el área de un triángulo que aproxime la forma que se muestra en la figura 11.19. Se tiene

$$\text{área} = \frac{1}{2}\,\text{base*altura}$$

$$\text{área} = 0.5*(5\text{-}1)*2400 = 4800$$



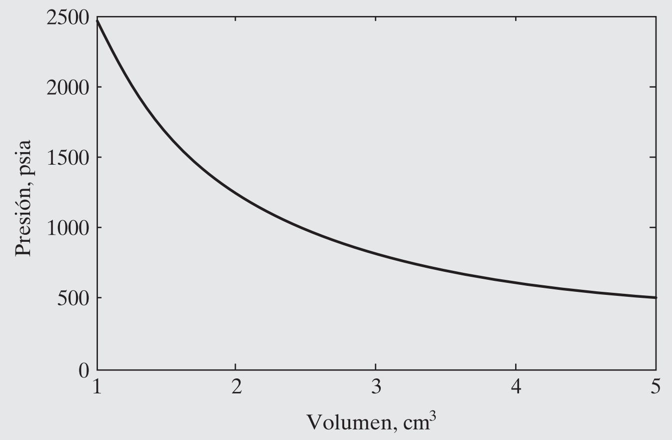


Figura 11.18
Para un sistema isotérmico, conforme el volumen aumenta, la presión disminuye.

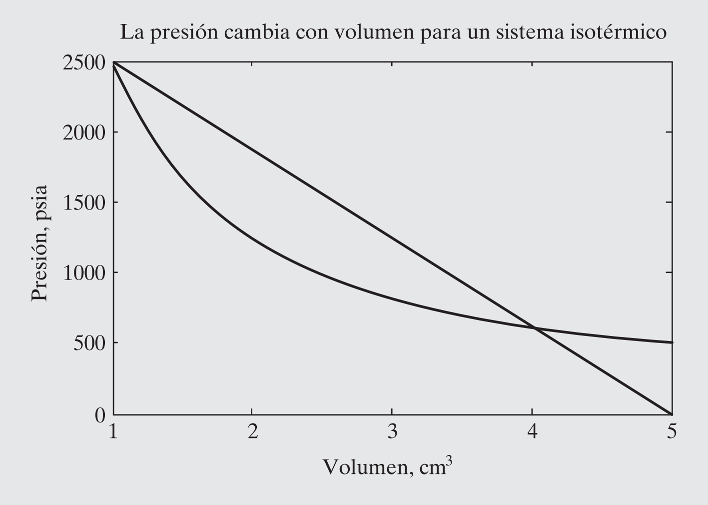


Figura 11.19
Se puede estimar el área bajo la curva con un triángulo.

que corresponde a 4800 kJ. Esto coincide bastante bien con el valor calculado de 4014 kJ.

Ahora que se tiene un proceso que funciona, se podría crear un archivo-m que conmine al usuario a ingresar valores para cualquier cambio en volumen:

```
clear,clc
syms P V n R T V1 V2                %Defina variables
ideal_gas_law=sym('P*V=n*R*T')      %Defina ley de gas ideal
P=solve(ideal_gas_law,'P')          %Resuelve para P
W=int(P,V,V1,V2)                    %Integre para encontrar trabajo

%Ahora permita al usuario ingresar los datos

temp=input('Ingrese una temperatura: ')
v1=input(' Ingrese el volumen inicial: ')
v2=input('Ingrese el volumen final: ')
work=subs(W,{n,R,V1,V2,T},{1,8.314,v1,v2,temp})
```

Este archivo-m genera la siguiente interacción con el usuario:

```
Ingrese una temperatura: 300
temp =
   300
Ingrese el volumen inicial: 1
v1 =
   1
Ingrese el volumen final: 5
v2 =
   5
work =
   4.0143e+003
```

11.5 ECUACIONES DIFERENCIALES

Las ecuaciones diferenciales contienen tanto variables dependientes como la derivada de la variable dependiente con respecto a la variable independiente. Por ejemplo,

$$\frac{dy}{dt} = y$$

es una ecuación diferencial.

Idea clave: la variable independiente por defecto para ecuaciones diferenciales en MATLAB es t.

Aunque cualquier símbolo se puede usar para la variable independiente o para la variable dependiente, la variable independiente por defecto en MATLAB es **t** (y es la elección usual para la mayoría de las formulaciones de ecuación diferencial ordinaria). Considere esta simple ecuación:

$$y = e^t$$

La derivada de y con respecto a t es

$$\frac{dy}{dt} = e^t$$

Esto también se podría expresar como una ecuación diferencial:

$$\frac{dy}{dt} = y$$

Por lo general, las ecuaciones diferenciales tienen más de una solución. La siguiente familia de funciones de t se podría expresar mediante la misma ecuación diferencial ($dy/dt = y$):

$$y = C_1 e^t$$

Puede especificar la ecuación particular de interés al especificar una condición inicial. Por ejemplo, si

$$y(0) = 1,$$

entonces

$$C_1 = 1$$

Una función ligeramente más complicada de y puede ser

$$y = t^2$$

La derivada de y con respecto a t es

$$\frac{dy}{dt} = 2t$$

Si se quiere, se podría rescribir esta ecuación como

$$\frac{dy}{dt} = \frac{2t^2}{t} = \frac{2y}{t}$$

La caja de herramientas simbólica incluye una función llamada **dsolve** que resuelve ecuaciones diferenciales. (Cuando se resuelve una ecuación diferencial, se busca una expresión para y en términos de t.) Esta función requiere que el usuario ingrese la ecuación diferencial, usando el símbolo **D** para especificar derivadas con respecto a la variable independiente, como en

```
dsolve('Dy=y')
ans =
C1*exp(t)
```

El uso de una sola entrada resulta en una familia de resultados. Si también incluye un segundo campo que especifique una condición inicial (o una condición de frontera), se regresa la respuesta exacta:

```
dsolve('Dy=y','y(0)=1')
ans =
exp(t)
```

De manera similar,

```
dsolve('Dy=2*y/t','y(-1)=1')
ans =
t^2
```

Si *t* no es la variable independiente en su ecuación diferencial, puede especificar la variable independiente en un tercer campo:

```
dsolve('Dy=2*y/t','y(-1)=1', 't')
ans =
t^2
```

Si una ecuación diferencial incluye sólo una primera derivada, se llama ecuación diferencial de primer orden. Las ecuaciones diferenciales de segundo orden incluyen una segunda derivada; las ecuaciones de tercer orden, una tercera derivada, etcétera. Para especificar una derivada de orden superior en la función **dsolve**, ponga el orden inmediatamente después de la **D**. Por ejemplo,

```
dsolve('D2y=-y')
ans =
C1*sin(t)+C2*cos(t)
```

resuelve una ecuación diferencial de segundo orden.

Sugerencia

No use la letra **D** en los nombres de sus variable en ecuaciones diferenciales. La función interpretará la **D** como especificación de una derivada.

La función **dsolve** también se puede usar para resolver sistemas de ecuaciones diferenciales. Primero mencione las ecuaciones a resolver, y luego las condiciones. La función **dsolve** aceptará hasta 12 entradas. Por ejemplo:

```
dsolve('eq1,eq2,...', 'cond1,cond2,...', 'v')
```

o

```
dsolve('eq1','eq2',...,'cond1','cond2',...,'v')
```

(La variable **v** es la variable independiente.) Ahora considere el siguiente ejemplo:

```
a=dsolve('Dx=y','Dy=x')
a =
  x: [1x1 sym]
  y: [1x1 sym]
```

Los resultados se reportan como elementos simbólicos en un arreglo estructura, tal como los resultados se reportaron con el comando **solve**. Para acceder a dichos elementos, use la sintaxis del arreglo estructura:

```
a.x
ans =
C1*exp(t)-C2*exp(-t)
```

y

```
a.y
ans =
C1*exp(t)+C2*exp(-t)
```

También podría especificar múltiples salidas desde la función:

```
[x,y]=dsolve('Dx=y','Dy=x')
x =
C1*exp(t)-C2*exp(-t)
y =
C1*exp(t)+C2*exp(-t)
```

Idea clave: no toda ecuación diferencial se puede resolver analíticamente.

MATLAB no puede resolver simbólicamente toda ecuación diferencial. Para sistemas de ecuaciones complicados (o mal condicionados), puede encontrar más fácil usar Maple. (Recuerde que la capacidad simbólica de MATLAB se basa en el motor Maple 8.) Existen muchas ecuaciones diferenciales que no se pueden resolver analíticamente en absoluto, sin importar cuán sofisticada sea la herramienta. Para dichas ecuaciones, con frecuencia son suficientes las técnicas numéricas.

RESUMEN

La caja de herramientas matemáticas simbólicas de MATLAB usa el software Maple 8, producido por Waterloo Maple. La caja de herramientas simbólica es un componente opcional de la versión profesional de MATLAB. Un subconjunto de la caja de herramientas simbólica, incluidos los componentes más populares, acompaña a la versión estudiantil. La sintaxis que usa la caja de herramientas simbólica es similar a la que usa Maple; sin embargo, debido a que la estructura subyacente de cada programa es diferente, los usuarios de Maple reconocerán algunas diferencias en la sintaxis.

Las variables simbólicas se crean en MATLAB con los comandos **sym** o **syms**:

```
x=sym('x') o
syms x
```

El comando **syms** tiene la ventaja de que hace fácil crear múltiples variables simbólicas en un enunciado:

```
syms a b c
```

El comando **sym** se puede usar para crear expresiones o ecuaciones completas en un solo paso:

```
y=sym('z^2-3')
```

Aunque **z** se incluye en esta expresión simbólica, no se define explícitamente como variable simbólica.

Una vez definidas las variables simbólicas, se pueden usar para crear expresiones más complicadas. Dado que **x**, **a**, **b** y **c** se definieron como variables simbólicas, se pueden combinar para crear la ecuación cuadrática:

```
EQ=a*x^2 + b*x + c
```

MATLAB permite a los usuarios manipular expresiones simbólicas o ecuaciones simbólicas. Las ecuaciones se igualan a algo; las expresiones no. Todos los enunciados en este resumen hasta el momento han creado expresiones. En contraste, el enunciado

```
EQ = sym('n=m/MW')
```

define una ecuación simbólica.

Tanto las expresiones simbólicas como las ecuaciones simbólicas se pueden manipular mediante funciones internas MATLAB de la caja de herramientas simbólica. La función **numden** extrae el numerador y denominador de una expresión, pero no es válido para ecuaciones. Las funciones **expand**, **factor** y **collect** se pueden usar para modificar una expresión o una ecuación. La función **simplify** simplifica una expresión o una ecuación sobre la base de reglas internas Maple, y la función **simple** prueba cada miembro de la familia de funciones de simplificación y reporta la respuesta más corta.

Una de las funciones simbólicas más útiles es **solve**, que permite al usuario resolver ecuaciones simbólicamente. Si la entrada a la función es una expresión, MATLAB iguala la expresión a cero. La función **solve** puede resolver no sólo una sola ecuación para la variable especificada, sino también sistemas de ecuaciones. A diferencia de las técnicas empleadas en álgebra matricial para resolver sistemas de ecuaciones, la entrada a **solve** no necesita ser lineal.

La función sustitución, **subs**, permite al usuario sustituir variables con valores numéricos o con nuevas variables. Es importante recordar que si una variable no se define explícitamente como simbólica, se debe encerrar en apóstrofes cuando se usa en la función **subs**. Cuando **y** se define como

```
y=sym('m +2*n + p')
```

las variables **m**, **n** y **p** no se definen explícitamente como simbólicas y, por tanto, se deben encerrar entre apóstrofes. Note que, cuando se sustituyen múltiples variables, se mencionan dentro de llaves. Si se sustituye una sola variable no se requieren las llaves. Dada la definición precedente de **y**, el comando

```
subs(y,{'m','n','p'}, {1,2,3})
```

regresa

```
ans =
   8
```

El comando **subs** se puede usar para sustituir tanto valores numéricos como variables simbólicas.

La capacidad de graficación simbólica de MATLAB refleja aproximadamente las opciones de graficación estándar. La más útil de estas gráficas para ingenieros y científicos probablemente es la gráfica *x-y*, **ezplot**. Esta función acepta una expresión simbólica y las grafica para valores de **x** desde -2π hasta $+2\pi$. El usuario también puede asignar los valores mínimo y máximo de **x**. las gráficas simbólicas se anotan con el uso de la misma sintaxis que las gráficas MATLAB estándar.

La caja de herramientas simbólica incluye algunas funciones de cálculo, de las cuales las más básicas son **diff** (diferenciación) e **int** (integración). La función **diff** permite al usuario obtener la derivada con respecto a una variable por defecto (**x** o cualquiera que esté más cerca de **x** en la expresión) o especificar la variable de diferenciación. También se pueden especificar derivadas de orden superior. La función **int** también permite al usuario integrar

con respecto a la variable por defecto (**x**) o especificar la variable de integración. Se pueden evaluar integrales definidas e indefinidas. Están disponibles funciones de cálculo adicionales no discutidas en este capítulo. Use la función **help** para más información.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Caracteres especiales	
`''`	identifica una variable simbólica que no se ha definido explícitamente
`{ }`	encierra un arreglo celda, usado en la función **solve** para crear listas de variables simbólicas

Comandos y funciones	
`collect`	recopila términos iguales
`diff`	encuentra la derivada simbólica de una expresión simbólica
`dsolve`	solucionador de ecuaciones diferenciales
`expand`	expande una expresión o ecuación
`ezcontour`	crea una gráfica de contorno
`ezcontourf`	crea una gráfica de contorno llena
`ezmesh`	crea una gráfica de malla a partir de una expresión simbólica
`ezmeshc`	grafica tanto malla como contorno creadas a partir de una expresión simbólica
`ezplot`	grafica una expresión simbólica (crea una gráfica *x*-*y*)
`ezplot3`	crea una gráfica lineal tridimensional
`ezpolar`	crea una gráfica en coordenadas polares
`ezsurf`	crea una gráfica de superficie a partir de una expresión simbólica
`ezsurfc`	grafica tanto malla como contorno creadas a partir de una expresión simbólica
`factor`	factoriza una expresión o ecuación
`int`	encuentra la integral simbólica de una expresión simbólica
`numden`	extrae el numerador y denominador de una expresión o una ecuación
`simple`	prueba y reporta sobre todas las funciones de simplificación y selecciona la respuesta más corta
`simplify`	simplifica con las reglas de simplificación internas de Maple
`solve`	resuelve una expresión o ecuación simbólica
`subs`	sustituye en una expresión o ecuación simbólica
`sym`	crea una variable, expresión o ecuación simbólica
`syms`	crea variables simbólicas

PROBLEMAS

Álgebra simbólica

11.1 Cree las variables simbólicas

```
a b c d x
```

y úselas para crear las siguientes expresiones simbólicas:

```
se1 = x^3 -3*x^2 +x
se2 = sin(x) + tan(x)
se3 =(2*x^2 - 3*x - 2)/(x^2 - 5*x)
se4 = (x^2 -9)/(x+3)
```

11.2 **(a)** Divida **se1** entre **se2**.
(b) Multiplique **se3** por **se4**.
(c) Divida **se1** entre **x**.
(d) Sume **se1** a **se3**.

11.3 Cree las siguientes ecuaciones simbólicas:

(a) `sq1=sym('x^2 + y^2 =4')`
(b) `sq2 = sym('5*x^5 - 4*x^4 + 3*x^3 + 2*x^2 -x =24 ')`
(c) `sq3 = sym('sin(a) + cos(b) -x*c = d')`
(d) `sq4 = sym('(x^3 - 3*x)/(3-x)=14')`

11.4 Intente usar la función **numden** para extraer el numerador y el denominador de **se4** y **sq4**. ¿Esta función también sirve para expresiones y ecuaciones? Describa cómo varían sus resultados. Intente explicar las diferencias.

11.5 Use las funciones **expand**, **factor**, **collect**, **simplifiy** y **simple** en **se1** a **se4**, y en **sq1** a **sq4**. En sus propias palabras, describa cómo operan estas funciones para los diversos tipos de ecuaciones y expresiones.

Resolución simbólica y uso del comando subs

11.6 Resuelva cada una de las expresiones creadas en el problema 11.1 para **x**.

11.7 Resuelva cada una de las ecuaciones creadas en el problema 11.3 para **x**.

11.8 Resuelva la ecuación **sq3**, creada en el problema 11.3, para **a**.

11.9 Un péndulo es un objeto rígido suspendido de un punto pivote sin fricción. (Véase la figura P11.9.) Si el péndulo se deja balancear con una inercia dada, se puede encontrar la frecuencia de oscilación con la ecuación

$$2\pi f = \sqrt{\frac{mgL}{I}}$$

donde

f = frecuencia,
m = masa del péndulo,
g = aceleración debida a la gravedad,
L = distancia desde el punto pivote al centro de gravedad del péndulo, e
I = inercia.

Use la capacidad simbólica de MATLAB para resolver la longitud L.

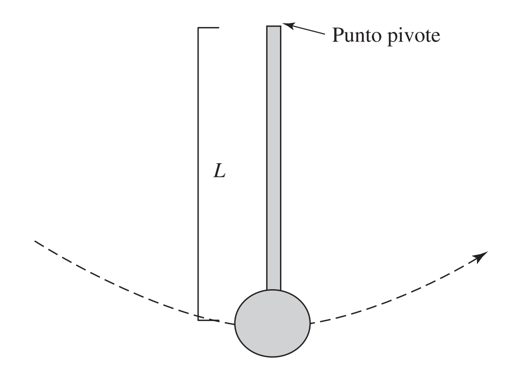


Figura P11.9
Péndulo descrito en el problema 11.9.

11.10 Sean la masa, inercia y frecuencia del péndulo en el problema anterior, respectivamente,

$$m = 10 \text{ kg},$$
$$f = 0.2 \text{ s}^{-1}, \text{ y}$$
$$I = 60 \text{ kg m/s}.$$

Si el péndulo está en la Tierra ($g = 9.8$ m/s^2), ¿cuál es la longitud desde el punto pivote hasta el centro de gravedad? (Use la función **subs** para resolver este problema.)

11.11 La energía cinética se define como

$$\text{EC} = \frac{1}{2} mV^2$$

donde
EC = energía cinética, medida en joules,
m = masa, medida en kg, y
V = velocidad, medida en m/s.
Cree una ecuación simbólica para energía cinética y resuélvala para velocidad.

11.12 Encuentre la energía cinética de un automóvil que pesa 2000 lb_m y viaja a 60 mph. (Véase la figura P11.12.) Sus unidades serán lb_m milla^2/h^2. Una vez que calcule este resultado, cámbielo a Btu al usar los siguientes factores de conversión:

$$1\ \text{lb}_f = 32.174\ \text{lbm} \cdot \text{ft/s}^2$$
$$1\ \text{h} = 3600\ \text{s}$$
$$1\ \text{milla} = 5280\ \text{ft}$$
$$1\ \text{Btu} = 778.169\ \text{ft} \cdot \text{lb}_f$$

11.13 La capacidad calorífica de un gas se puede modelar con la siguiente ecuación, compuesta de las constantes empíricas a, b, c y d y la temperatura T en grados K:

$$C_P = a + bT + cT^2 + dT^3$$

Las constantes empíricas no tienen un significado físico, pero se usan para hacer que la ecuación ajuste los datos. Cree una ecuación simbólica para capacidad calorífica y resuelva para T.

11.14 Sustituya los siguientes valores para a, b, c y d en la ecuación de capacidad calorífica del problema anterior y dé a su resultado un nuevo nombre (estos valores modelan la capacidad calorífica del nitrógeno gaseoso en kJ/(kmol K) conforme cambia temperatura entre aproximadamente 273 y 1800 K):

$$a = 28.90$$
$$b = -0.1571 \times 10^{-2}$$
$$c = 0.8081 \times 10^{-5}$$
$$d = -2.873 \times 10^{-9}$$



Figura P11.12
Auto descrito en el problema 11.12.

Resuelva su nueva ecuación para T si la capacidad calorífica (C_p) es igual a 29.15 kJ/(kmol K).

11.15 La ecuación Antoine usa constantes empíricas para modelar la presión de vapor de un gas como función de la temperatura. La ecuación modelo es

$$\log_{10}(P) = A - \frac{B}{C + T}$$

donde

P	=	presión, en mmHg,
A	=	constante empírica,
B	=	constante empírica,
C	=	constante empírica, y
T	=	temperatura en grados C.

El punto de ebullición normal de un líquido es la temperatura a la que la presión de vapor (P) del gas es igual a la presión atmosférica, 760 mmHg. Use la capacidad simbólica MATLAB para encontrar el punto de ebullición normal del benceno si las constantes empíricas son

$$A = 6.89272$$
$$B = 1203.531$$
$$C = 219.888$$

11.16 Un estudiante hambriento va a la cafetería y compra su almuerzo. Al día siguiente gasta el doble de dinero. Al tercer día gasta \$1 menos que el segundo día. Al final de los tres días gasta \$35. ¿Cuánto gastó cada día? Use la capacidad simbólica de MATLAB para ayudar a resolver este problema.

Resolución de sistemas de ecuaciones

11.17 Considere el siguiente conjunto de siete ecuaciones:

$$\begin{aligned}
3x_1 + 4x_2 + 2x_3 - x_4 + x_5 + 7x_6 + x_7 &= 42\\
2x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 + 2x_6 + 8x_7 &= 32\\
x_1 + 2x_2 + 3x_3 + x_4 + 2x_5 + 4x_6 + 6x_7 &= 12\\
5x_1 + 10x_2 + 4x_3 + 3x_4 + 9x_5 - 2x_6 + x_7 &= -5\\
3x_1 + 2x_2 - 2x_3 - 4x_4 - 5x_5 - 6x_6 + 7x_7 &= 10\\
-2x_1 + 9x_2 + x_3 + 3x_4 - 3x_5 + 5x_6 + x_7 &= 18\\
x_1 - 2x_2 - 8x_3 + 4x_4 + 2x_5 + 4x_6 + 5x_7 &= 17
\end{aligned}$$

Defina una variable simbólica para cada una de las ecuaciones y use la capacidad simbólica de MATLAB para resolver cada una de las incógnitas.

11.18 Compare la cantidad de tiempo que le toma resolver el problema anterior si usa división izquierda y matemática simbólica con las funciones **tic** y **toc**, cuya sintaxis es

```
tic
⋮
código a cronometrar
⋮
toc
```

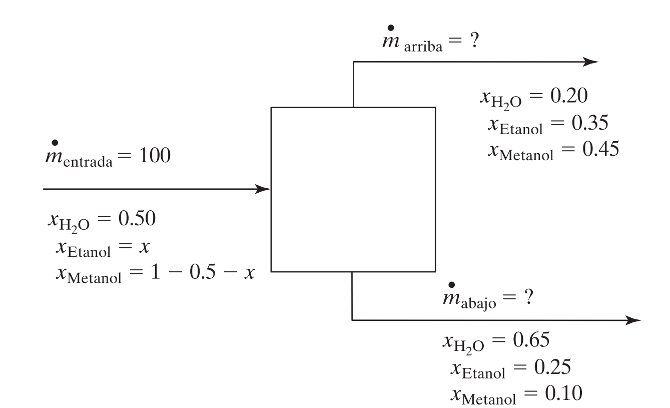


Figura P11.19
Proceso de separación con tres componentes, descrito en el problema 11.19.

11.19 Use las capacidades simbólicas de MATLAB para resolver el siguiente problema mediante álgebra matricial:

Considere un proceso de separación en el que una corriente de agua, etanol y metanol entra a una unidad de proceso. Dos corrientes salen de la unidad, cada una con cantidades variables de los tres componentes. (Véase la figura P11.19.)

Determine las tasas de flujo de masa adentro del sistema y afuera por arriba y abajo de la unidad de separación.

(a) Primero configure las siguientes ecuaciones de balance de materiales para cada uno de los tres componentes:

Agua

$$0.5(100) = 0.2m_{arriba} + 0.65m_{abajo}$$

$$50 = 0.2m_{arriba} + 0.65m_{abajo}$$

Etanol

$$100x = 0.35m_{arriba} + 0.25m_{abajo}$$

$$0 = -100x + 0.35m_{arriba} + 0.25m_{abajo}$$

Metanol

$$100(1 - 0.5 - x) = 0.45m_{arriba} + 0.1m_{abajo}$$

$$50 = 100x + 0.45m_{arriba} + 0.1m_{abajo}$$

(b) Cree ecuaciones simbólicas para representar cada balance de material.
(c) Use la función **solve** para resolver el sistema de tres ecuaciones y tres incógnitas.

11.20 Considere las siguientes dos ecuaciones:

$$x^2 + y^2 = 42$$

$$x + 3y + 2y^2 = 6$$

Defina una ecuación simbólica para cada una y resuélvala con la capacidad simbólica de MATLAB. ¿Podría resolver estas ecuaciones usando matrices? Intente este problema dos veces, una vez sólo con enteros en sus definiciones de ecuación y una vez con números punto flotante (los que tienen puntos decimales). ¿Cómo varían sus resultados? Verifique la ventana del área de trabajo para determinar si los resultados todavía son simbólicos.

Graficación simbólica

11.21 Cree gráficas de las siguientes expresiones desde $x = 0$ hasta 10:

(a) $y = e^x$

(b) $y = \text{sen}(x)$

(c) $y = ax^2 + bx + c$, donde $a = 5, b = 2$ y $c = 4$

(d) $y = \sqrt{x}$

Cada una de sus gráficas debe incluir un título, una etiqueta de eje x, una etiqueta de eje y y una retícula.

11.22 Use **ezplot** para graficar las siguientes expresiones en la misma figura para valores x de -2π a 2π (necesitará usar el comando **hold on**):

$$y_1 = \text{sen}(x)$$
$$y_2 = \text{sen}(2x)$$
$$y_3 = \text{sen}(3x)$$

Use las herramientas de graficación interactivas para asignar a cada línea un color y estilo de línea diferente.

11.23 Use **ezplot** para graficar las siguientes ecuaciones implícitas:

(a) $x^2 + y^3 = 0$

(b) $x + x^2 - y = 0$

(c) $x^2 + 3y^2 = 3$

(d) $x \cdot y = 4$

11.24 Use **ezplot** para graficar las siguientes funciones paramétricas:

(a) $f_1(t) = x = \text{sen}(t)$
$f_2(t) = y = \cos(t)$

(b) $f_1(t) = x = \text{sen}(t)$
$f_2(t) = y = 3\cos(t)$

(c) $f_1(t) = x = \text{sen}(t)$
$f_2(t) = y = \cos(3t)$

(d) $f_1(t) = x = 10\,\text{sen}(t)$
$f_2(t) = y = t\cos(t)$ de $t = 0$ a 30

(e) $f_1(t) = x = t\,\text{sen}(t)$
$f_2(t) = y = t\cos(t)$ de $t = 0$ a 30

11.25 La distancia que recorre un proyectil cuando se dispara a un ángulo θ es una función del tiempo y se puede dividir en distancias horizontal y vertical (véase la figura P11.25), dados respectivamente por

$$\text{Horizontal}(t) = tV_0\cos(\theta)$$

y

$$\text{Vertical}(t) = tV_0\,\text{sen}(\theta) - \frac{1}{2}gt^2$$

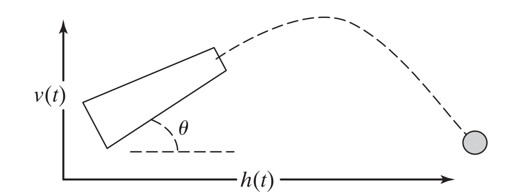


Figura P11.25
Trayectoria de un proyectil.

donde

Horizontal	=	distancia recorrida en la dirección x,
Vertical	=	distancia recorrida en la dirección y,
V_0	=	velocidad inicial del proyectil,
g	=	aceleración debida a la gravedad, 9.8 m/s^2, y
t	=	tiempo, s.

Suponga que un proyectil se dispara con una velocidad inicial de 100 m/s y un ángulo de lanzamiento de $\pi/4$ radianes (45°). Use **ezplot** para graficar distancia horizontal en el eje x y distancia vertical en el eje y para tiempos de 0 a 20 segundos.

11.26 Para cada una de las siguientes expresiones, use la función gráfica **ezpolar** para crear una gráfica de la expresión, y use la función **subplot** para poner sus cuatro gráficas en la misma figura:

(a) $\text{sen}^2(\theta) + \cos^2(\theta)$
(b) $\text{sen}(\theta)$
(c) $e^{\theta/5}$ para θ de 0 a 20
(d) $\text{senh}(\theta)$ para θ de 0 a 20

11.27 Use **ezplot3** para crear una gráfica lineal tridimensional de las funciones siguientes:

$$f_1(t) = x = t\,\text{sen}(t)$$
$$f_2(t) = y = t\cos(t)$$
$$f_3(t) = z = t$$

11.28 Use la siguiente ecuación para crear una función simbólica **Z**:

$$Z = \frac{\text{sen}\left(\sqrt{X^2 + Y^2}\right)}{\sqrt{X^2 + Y^2}}$$

(a) Use la función de graficación **ezmesh** para crear una gráfica tridimensional de **Z**.
(b) Use la función de graficación **ezsurf** para crear una gráfica tridimensional de **Z**.
(c) Use **ezcontour** para crear un mapa de contorno de **Z**.
(d) Genere una combinación de superficie y contorno de **Z**, con **ezsurfc**.

Use subgráficas para poner todas las gráficas que crea en la misma figura.

Cálculo

11.29 Determine la primera y segunda derivadas de las siguientes funciones, use las funciones simbólicas MATLAB:

(a) $f_1(x) = y = x^3 - 4x^2 + 3x + 8$
(b) $f_2(x) = y = (x^2 - 2x + 1)(x - 1)$

(c) $f_3(x) = y = \cos(2x)\,\text{sen}(x)$

(d) $f_4(x) = y = 3xe^{4x^2}$

11.30 Use las funciones simbólicas de MATLAB para realizar las integraciones siguientes:

(a) $\int (x^2 + x)\,dx$

(b) $\int_{0.3}^{1.3} (x^2 + x)\,dx$

(c) $\int (x^2 + y^2)\,dx$

(d) $\int_{3.5}^{24} (ax^2 + bx + c)\,dx$

11.31 Sea el siguiente polinomio que representa la altitud en metros durante las primeras 48 horas siguientes al lanzamiento de un globo meteorológico:

$$h(t) = -0.12t^4 + 12t^3 - 380t^2 + 4100t + 220$$

Suponga que las unidades de t son horas.

(a) Use MATLAB junto con el hecho de que la velocidad es la primera derivada de la altitud para determinar la ecuación para la velocidad del globo.
(b) Use MATLAB junto con el hecho de que la aceleración es la derivada de la velocidad, o la segunda derivada de la altitud, para determinar la ecuación para la aceleración del globo.
(c) Use MATLAB para determinar cuándo el globo golpea el suelo. Puesto que $h(t)$ es un polinomio de cuarto orden, habrá cuatro respuestas. Sin embargo, sólo una respuesta será físicamente significativa.
(d) Use la capacidad de graficación simbólica de MATLAB para crear gráficas de altitud, velocidad y aceleración de tiempo 0 hasta que el globo golpea el suelo [que se determinó en la parte (c)]. Necesitará tres gráficas separadas, pues altitud, velocidad y aceleración tienen unidades diferentes.
(e) Determine la altura máxima que alcanza el globo. Use el hecho de que la velocidad del globo es cero a la altura máxima.

11.32 Suponga que se bombeará agua en un tanque inicialmente vacío. (Véase la figura P11.32.) Se sabe que la tasa de flujo de agua en el tanque en el tiempo t (en segundos) es $50 - t$ litros por segundo. Se puede demostrar que la cantidad de agua Q que fluye en el tanque durante los primeros x segundos es igual a la integral de la expresión $(50 - t)$ evaluada de 0 a x segundos.*

(a) Determine una ecuación simbólica que represente la cantidad de agua en el tanque después de x segundos.
(b) Determine la cantidad de agua en el tanque después de 30 segundos.
(c) Determine la cantidad de agua que fluyó en el tanque entre los segundos 10 y 15 después de iniciado el flujo.

---

*Tomado de Etter, Kuncicky y Moore, *Introduction to MATLAB 7* (Upper Saddle River, NJ: Pearson/Prentice Hall, 2005).

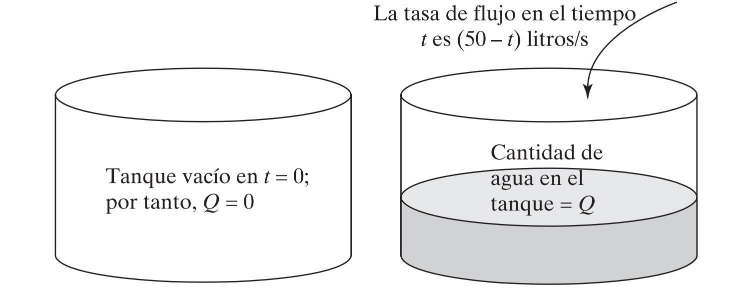


Figura P11.32
Problema de llenado de tanque.

11.33 Considere un resorte con el extremo izquierdo fijo y el derecho libre para moverse a lo largo del eje x. (Véase la figura P11.33.) Se supone que el extremo derecho del resorte está en el origen $x = 0$ cuando el resorte está en reposo. Cuando el resorte se estira, el extremo derecho del resorte está en un nuevo valor de x que es mayor que cero. Cuando el resorte se comprime, el extremo derecho del resorte está en algún valor que es menor que cero. Suponga que el resorte tiene una longitud natural de 1 ft y que se requiere una fuerza de 10 lb para comprimir el resorte a una longitud de 0.5 ft. Entonces se puede demostrar que el trabajo, en ft lb_f, realizado para estirar el resorte desde su longitud natural hasta un total de n ft es igual a la integral de $20x$ sobre el intervalo de 0 a $n - 1$.*

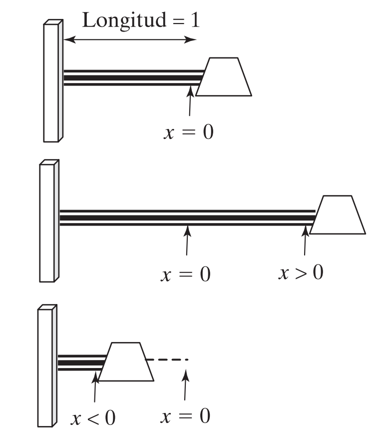


Figura P11.33
Problema del resorte descrito en el problema 11.33.

(a) Use MATLAB para determinar una expresión simbólica que represente la cantidad de trabajo necesario para estirar el resorte a una longitud total de n ft.
(b) ¿Cuál es la cantidad de trabajo realizado para estirar el resorte a un total de 2 ft?
(c) Si la cantidad de trabajo ejercido es 25 ft lb_f, ¿cuál es la longitud del resorte estirado?

11.34 La capacidad calorífica C_p de un gas se puede modelar con la ecuación empírica

$$C_p = a + bT + cT^2 + dT^3$$

*Tomado de Etter, Kuncicky y Moore, *Introduction to MATLAB 7* (Upper Saddle River, NJ: Pearson/Prentice Hall, 2005).

donde a, b, c y d son constantes empíricas y T es la temperatura en grados Kelvin. El cambio en entalpía (una medida de energía) conforme el gas se caliente de T_1 a T_2 es la integral de esta ecuación con respecto a T:

$$\Delta h = \int_{T_1}^{T_2} C_p \, dT$$

Encuentre el cambio en entalpía del oxígeno gaseoso conforme se calienta de 300 K a 1000 K. Los valores de a, b, c y d para el oxígeno son

$$a = 25.48$$
$$b = 1.520 \times 10^{-2}$$
$$c = -0.7155 \times 10^{-5}$$
$$d = 1.312 \times 10^{-9}$$



CAPÍTULO 12

Técnicas numéricas

Objetivos

Después de leer este capítulo, el alumno será capaz de

- interpolar entre puntos de datos, con modelos lineales o cúbicos segmentarios (spline).
- modelar un conjunto de puntos de datos como un polinomio.
- usar la herramienta de ajuste básico.
- usar la caja de herramientas de ajuste de curvas.
- realizar diferenciaciones numéricas.
- realizar integraciones numéricas.
- resolver numéricamente ecuaciones diferenciales.

12.1 INTERPOLACIÓN

Cuando se miden cosas, comúnmente no se recopilan datos en todo posible punto de datos. Considere un conjunto de datos *xy* recopilados durante un experimento. Al usar una técnica de interpolación, se puede estimar el valor de *y* en valores de *x* donde no se realiza una medición. (Véase la figura 12.1.) Las dos técnicas de interpolación más comunes son la interpolación lineal y la interpolación cúbica segmentaria (spline o de trazador), las cuales son soportadas por MATLAB.

12.1.1 Interpolación lineal

La forma más común de estimar un punto de datos entre dos puntos conocidos es la *interpolación lineal*. En esta técnica, se supone que la función entre los puntos se puede estimar mediante una línea recta dibujada entre ellos, como se muestra en la figura 12.2. Si se encuentra la ecuación de una línea recta definida por los dos puntos conocidos, se puede encontrar *y* para cualquier valor de *x*. Cuanto más cerca estén los puntos, es más probable que sea más precisa la aproximación.

Sugerencia

Aunque posible, rara vez es aconsejable *extrapolar* la región donde se recopilaron los datos. Puede ser tentador suponer que los datos seguirán el mismo patrón, pero esta suposición puede conducir a grandes errores.

Sugerencia

El último carácter en el nombre de función **interp1** es el número uno. Dependiendo de la fuente, puede parecer como la letra ‘l’.

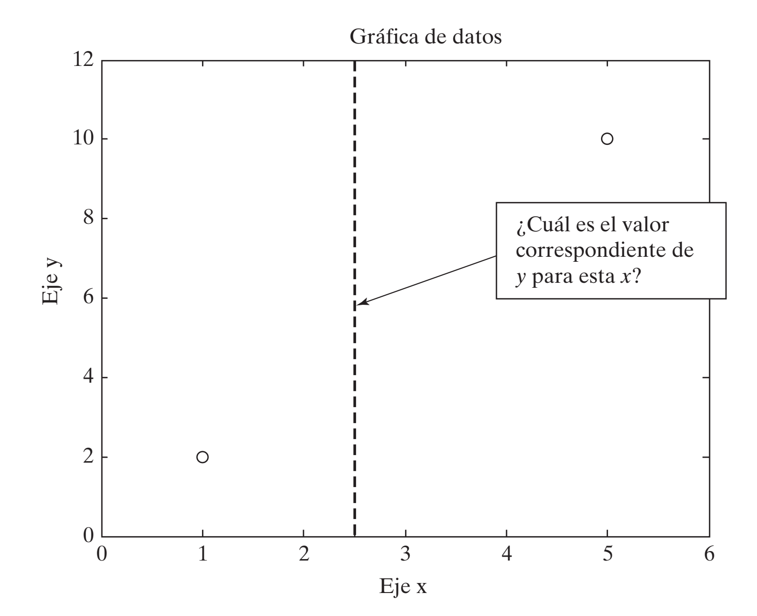


Figura 12.1
Interpolación entre puntos de datos.

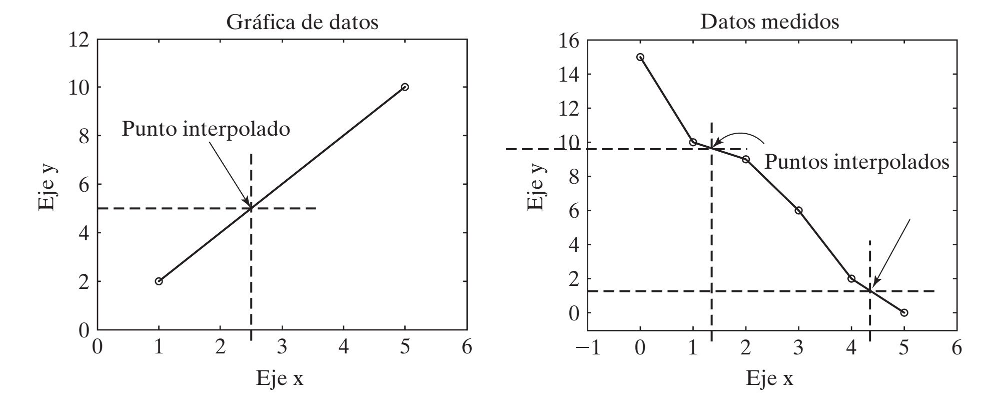


Figura 12.2
Interpolación lineal: conecte los puntos con una línea recta para encontrar *y*.

La interpolación lineal se puede realizar en MATLAB con la función **interp1**. Para emplear **interp1**, primero necesitará crear un conjunto de pares ordenados a usar como entrada para la función. Los datos que se usan para crear la gráfica de la derecha en la figura 12.2 son

```
x=0:5;
y=[15, 10, 9, 6, 2, 0];
```

Para realizar una sola interpolación, la entrada a **interp1** son los datos **x**, los datos **y** y el nuevo valor **x** para el que le gustaría estimar **y**. Por ejemplo, para estimar el valor de **y** cuando **x** es igual a 3.5, escriba

```
interp1(x,y,3.5)
ans =
   4
```

Puede realizar múltiples interpolaciones al mismo tiempo al colocar un vector de valores **x** en el tercer campo de la función **interp1**. Por ejemplo, para estimar valores **y** para nuevas **x** igualmente espaciadas desde 0 hasta 5 por 0.2, escriba

interpolación: técnica para estimar un valor intermedio con base en valores cercanos

```
new_x=0:0.2:5;
new_y=interp1(x,y,new_x)
```

que regresa

```
new_y =
 Columns 1 through 5
 15.0000 14.0000 13.0000 12.0000 11.0000
 Columns 6 through 10
 10.0000 9.8000 9.6000 9.4000 9.2000
 Columns 11 through 15
 9.0000 8.4000 7.8000 7.2000 6.6000
 Columns 16 through 20
 6.0000 5.2000 4.4000 3.6000 2.8000
 Columns 21 through 25
 2.0000 1.6000 1.2000 0.8000 0.4000
 Column 26
    0
```

Se puede presentar los resultados en la misma gráfica con los datos originales en la figura 12.3:

```
plot(x,y,new_x,new_y,'o')
```

(En este capítulo, por claridad, se dejaron fuera los comandos usados para agregar títulos y etiquetas de eje a las gráficas.)

La función **interp1** tiene por defecto la interpolación lineal para realizar sus estimaciones. Sin embargo, como se verá en la siguiente sección, son posibles otros enfoques. Si se

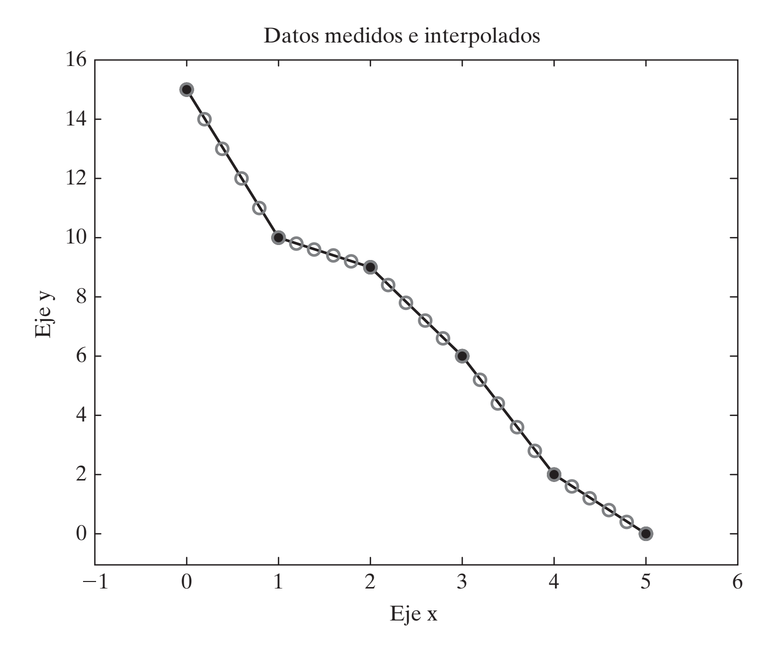


Figura 12.3
Tanto los datos medidos como los interpolados se mostraron en la misma gráfica. Los puntos originales se modificaron en la función de graficación interactiva para hacerlos círculos sólidos.

quiere (probablemente para propósitos de documentación) definir explícitamente el enfoque usado en **interp1** como interpolación lineal, se puede especificar en un cuarto campo:

```
interp1(x, y, 3.5, 'linear')
ans =
  4
```

12.1.2 Interpolación cúbica segmentaria

Conectar los puntos de datos con líneas rectas probablemente no es la mejor forma de estimar valores intermedios, aunque seguramente es la más simple. Se puede crear una curva más suave al usar la técnica de interpolación cúbica segmentaria (de trazador o spline), incluida en la función **interp1**. Este enfoque usa un polinomio de tercer orden para modelar el comportamiento de los datos. Para llamar la spline cúbica, se necesita agregar un cuarto campo a **interp1**:

```
interp1(x,y,3.5,'spline')
```

Este comando regresa una estimación mejorada de **y** en **x** = 3.5:

```
ans =
  3.9417
```

Desde luego, también se podría usar la técnica cúbica segmentaria para crear un arreglo de nuevas estimaciones para **y**, para cada miembro de un arreglo de valores **x**:

```
new_x=0:0.2:5;
new_y_spline=interp1(x,y,new_x,'spline');
```

Una gráfica de estos datos en la misma gráfica de los datos medidos (figura 12.4) con el uso del comando

```
plot(x,y,new_x,new_y_spline,'-o')
```

resulta en dos líneas diferentes.

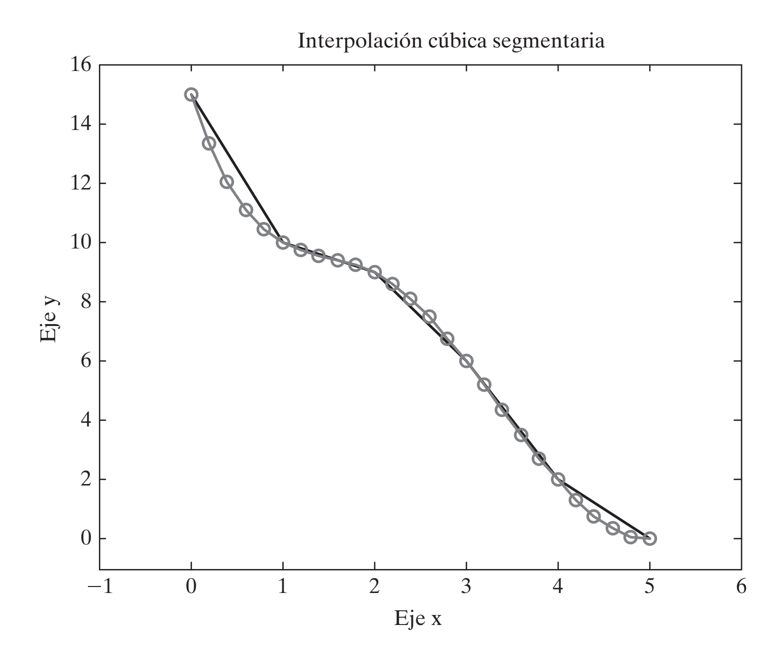


Figura 12.4
Interpolación cúbica segmentaria (de trazador o spline). Los puntos de datos en la curva suave se calcularon. Los puntos de datos en los segmentos de línea recta se midieron. Note que todo punto medido también cae en la línea curva.

Tabla 12.1 Opciones de interpolación en la función interp1

'linear'	interpolación lineal, que es por defecto	**interp1(x,y,3.5,'linear')** **ans =** **4**
'nearest'	interpolación al vecino más cercano	**interp1(x,y,3.5,'nearest')** **ans =** **2**
'spline'	interpolación cúbica segmentaria	**interp1(x,y,3.5,'spline')** **ans =** **3.9417**
'pchip'	interpolación cúbica segmentaria que preserva la forma	**interp1(x,y,3.5,'pchip')** **ans =** **3.9048**
'cubic'	igual que **'pchip'**	**interp1(x,y,3.5,'cubic')** **ans =** **3.9048**
'v5cubic'	la interpolación cúbica de MTALAB 5 que no extrapola y usa **'spline'** si **x** no está igualmente espaciada	**interp1(x,y,3.5,'v5cubic')** **ans =** **3.9375**

La línea curva en la figura 12.4 se dibujó con el uso de los puntos de datos interpolados. La línea compuesta de segmentos de línea recta se dibujó a través de los datos originales.

Aunque las formas más comunes de interpolar entre puntos de datos son los enfoques lineal y spline, MATLAB ofrece algunas otras opciones, como se menciona en la tabla 12.1.

EJEMPLO 12.1

Propiedades termodinámicas: uso de las tablas de vapor

La materia de termodinámica hace uso extenso de tablas. Aunque muchas propiedades termodinámicas se pueden describir mediante ecuaciones bastante simples, otras son pobremente entendidas o las ecuaciones que describen su comportamiento son muy complicadas. Es mucho más fácil sólo tabular los valores. Por ejemplo, considere los valores en la tabla 12.2 para vapor a 0.1 MPa (aproximadamente 1 atm) (figura 12.5).

Use interpolación lineal para determinar la energía interna a 215 °C. Use interpolación lineal para determinar la temperatura si la energía interna es 2600 kJ/kg.

1. Establezca el problema.

   Encontrar la energía interna del vapor con interpolación lineal.

   Encontrar la temperatura del vapor con interpolación lineal.

2. Describa las entradas y salidas.

   Entrada Tabla de temperatura y energía interna
   u desconocida
   T desconocida

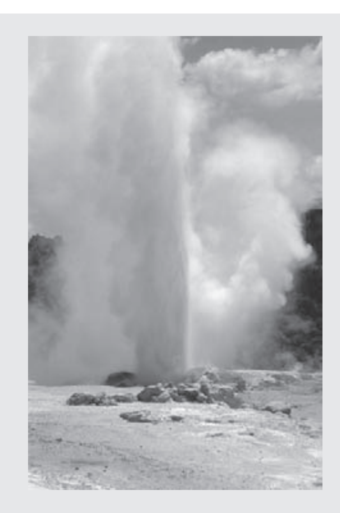


Figura 12.5
Los géiseres rocían agua y vapor a alta temperatura y gran presión.

Tabla 12.2 Energía interna como función de la temperatura

Temperatura, °C	Energía interna u, kJ/kg
100	2506.7
150	2582.8
200	2658.1
250	2733.7
300	2810.4
400	2967.9
500	3131.6

Datos tomados de Joseph H. Keenan, Frederick G. Keyes, Philip G. Hill, y Joan G. Moore, *Steam Tables, SI units* (New York: John Wiley and Sons, 1978).

Salida Energía interna
Temperatura

3. Desarrolle un ejemplo a mano.
En la primera parte del problema, se necesita encontrar la energía interna a 215 °C. La tabla incluye valores a 200 °C y 250 °C. Primero se necesita determinar la fracción de la distancia entre 200 y 250 a la que cae el valor 215:

$$\frac{215 - 200}{250 - 200} = 0.30$$

Si se modela la relación entre temperatura y energía interna como lineal, la energía interna también debe estar a 30% la distancia entre los valores tabulados:

$$0.30 = \frac{u - 2658.1}{2733.7 - 2658.1}$$

Al resolver para *u* se obtiene

$$u = 2680.78 \text{ kJ/kg}$$

4. Desarrolle una solución MATLAB.
   Cree la solución MATLAB en un archivo-m y luego córralo en el entorno de comando:

```
%Ejemplo 12.1
%Termodinámica
T=[100, 150, 200, 250, 300, 400, 500];
u= [2506.7, 2582.8, 2658.1, 2733.7, 2810.4, 2967.9, 3131.6];
newu=interp1(T,u,215)
newT=interp1(u,T,2600)
```

   El código regresa

```
newu =
   2680.78
newT =
   161.42
```

5. Ponga a prueba la solución.
   El resultado MATLAB coincide con el resultado a mano. Este enfoque se podría usar para cualquiera de las propiedades tabuladas en las tablas de vapor. Las tablas JANAF son una fuente similar de propiedades termodinámicas publicadas por el National Institute of Standards and Technology (Instituto Nacional de Estándares y Tecnología).

EJEMPLO 12.2

Propiedades termodinámicas: expansión de las tablas de vapor

Como se vio en el ejemplo 12.1, la termodinámica hace uso extenso de tablas. Comúnmente, muchos experimentos se llevan a cabo bajo presión atmosférica, de modo que regularmente tendrá necesidad de usar la tabla 12.3, que sólo es una porción de las tablas de vapor (figura 12.6).



Figura 12.6
Las plantas de electricidad usan vapor como un "fluido de trabajo".

Tabla 12.3 Propiedades de vapor supercalentado a 0.1 MPa (aproximadamente 1 atm)

Temperatura, °C	Volumen específico v, m^3/kg	Energía interna u, kJ/kg	Entalpía h, kJ/kg
100	1.6958	2506.7	2676.2
150	1.9364	2582.8	2776.4
200	2.172	2658.1	2875.3
250	2.406	2733.7	2974.3
300	2.639	2810.4	3074.3
400	3.103	2967.9	3278.2
500	3.565	3131.6	3488.1

Datos tomados de Joseph H. Keenan, Frederick G. Keyes, Philip G. Hill, y Joan G. Moore, *Steam Tables, SI units* (New York: John Wiley and Sons, 1978).

Note que la tabla está espaciada a intervalos de 50 grados al principio y luego a intervalos de 100 grados. Suponga que tiene un proyecto que le requiere el uso de esta tabla y usted preferiría no tener que realizar una interpolación lineal cada vez que la utilice. Use MATLAB para crear una tabla, a través de interpolación lineal, con un espaciamiento de temperatura de 25 grados.

1. Establezca el problema.
   Encontrar el volumen específico, energía interna y entalpía cada 5 grados.
2. Describa las entradas y salidas.

   Entrada Tabla de temperatura y energía interna
   Nuevo intervalo de tabla de 5 grados

   Salida Tabla
3. Desarrolle un ejemplo a mano.
   En el ejemplo 12.1 se encontró la energía interna a 215 °C. Dado que 215 no está en la tabla de salida, se volverán a hacer los cálculos a 225 °C:

$$\frac{225 - 200}{250 - 200} = 0.50$$

y

$$0.50 = \frac{u - 2658.1}{2733.7 - 2658.1}$$

Al resolver para u se obtiene

$$u = 2695.9 \text{ kJ/kg}$$

Se puede usar este mismo cálculo para confirmar los de la tabla que cree.

4. Desarrolle una solución MATLAB.
   Cree la solución MATLAB en un archivo-m y luego córralo en el entorno de comandos:

```
%Ejemplo 12.2
%Termodinámica
clear, clc
T=[100, 150, 200, 250, 300, 400, 500]';
v=[1.6958, 1.9364, 2.172, 2.406, 2.639, 3.103, 3.565]';
u=[2506.7, 2582.8, 2658.1, 2733.7, 2810.4, 2967.9, 3131.6]';
h=[2676.2, 2776.4, 2875.3, 2974.3, 3074.3, 3278.2, 3488.1]';
props=[v,u,h];
newT=[100:25:500]';
newprop=interp1(T,props,newT);
disp('Propiedades de vapor a 0.1 MPa')
disp('Temp Specific Volume Internal Energy Enthalpy')
disp(' C  m^3/kg kJ/kg kJ/kg')
fprintf('%6.0f %10.4f %8.1f %8.1f \n',[newT,newprop]')
```

   El código MATLAB imprime la tabla siguiente:

```
Propiedades de vapor a 0.1 MPa
Temp Specific Volume Internal Energy Enthalpy
C       m^3/kg      kJ/kg     kJ/kg
 100      1.6958    2506.7    2676.2
 125      1.8161    2544.8    2726.3
 150      1.9364    2582.8    2776.4
 175      2.0542    2620.4    2825.9
 200      2.1720    2658.1    2875.3
 225      2.2890    2695.9    2924.8
 250      2.4060    2733.7    2974.3
 275      2.5225    2772.1    3024.3
 300      2.6390    2810.4    3074.3
 325      2.7550    2849.8    3125.3
 350      2.8710    2889.2    3176.3
 375      2.9870    2928.5    3227.2
 400      3.1030    2967.9    3278.2
 425      3.2185    3008.8    3330.7
 450      3.3340    3049.8    3383.1
 475      3.4495    3090.7    3435.6
 500      3.5650    3131.6    3488.1
```

5. Ponga a prueba la solución.
   El resultado MATLAB coincide con el resultado a mano. Ahora que se sabe que el programa funciona, puede crear tablas más extensas al cambiar la definición de **newT** de

```
newT=[100:25:500]';
```

   a un vector con un incremento de temperatura más pequeño, por ejemplo

```
newT=[100:1:500]';
```

Ejercicio de práctica 12.1

Cree vectores *x* y *y* para representar los siguientes datos:

x	*y*
10	23
20	45
30	60
40	82
50	111
60	140
70	167
80	198
90	200
100	220

1. Grafique los datos en una gráfica *xy*.
2. Use interpolación lineal para aproximar el valor de *y* cuando $x = 15$.
3. Use interpolación cúbica segmentaria (spline) para aproximar el valor de *y* cuando $x = 15$.
4. Use interpolación lineal para aproximar el valor de *x* cuando $y = 80$.
5. Use interpolación cúbica segmentaria (spline) para aproximar el valor de *x* cuando $y = 80$.
6. Use interpolación cúbica segmentaria (spline) para aproximar valores *y* para valores *x* igualmente espaciados entre 10 y 100 a intervalos de 2.
7. Grafique los datos originales en una gráfica *xy* como puntos de datos no conectados por una línea. Además, grafique los valores calculados en el problema 6.

12.1.3 Interpolación multidimensional

Imagine que tiene un conjunto de datos *z* que depende de dos variables *x* y *y*. Por ejemplo, considere esta tabla:

	x = 1	**x = 2**	**x = 3**	**x = 4**
y = 2	7	15	22	30
y = 4	54	109	164	218
y = 6	403	807	1210	1614

Si quiere determinar el valor de *z* en $y = 3$ y $x = 1.5$, tendría que realizar dos interpolaciones. Un enfoque sería encontrar los valores de *z* en $y = 3$ y todos los valores *x* dados con el uso de **interp1** y luego hacer una segunda interpolación en su nuevo gráfico. Primero defina *x*, *y* y *z* en MATLAB:

```
y=2:2:6;
x=1:4;
z=[ 7   15  22   30
    54  109 164  218
    403 807 1210 1614];
```

Ahora se puede usar **interp1** para encontrar los valores de z en $y = 3$ para todos los valores x:

```
new_z=interp1(y,z,3)    regresa
new_z =
    30.50   62.00   93.00   124.00
```

Finalmente, dado que se tienen valores z en $y = 3$, se puede usar **interp1** para encontrar z en $y = 3$ y $x = 1.5$:

```
new_z2=interp1(x,new_z,1.5)
new_z2 =
    46.25
```

Aunque este enfoque funciona, es complicado tener que realizar los cálculos en dos pasos. MATLAB incluye una función de interpolación lineal bidimensional, **interp2**, que puede resolver el problema en un solo paso:

```
interp2(x,y,z,1.5,3)
ans =
 46.2500
```

El primer campo en la función **interp2** debe ser un vector que defina el valor asociado con cada columna (en este caso, **x**), y el segundo campo debe ser un vector que defina los valores asociados con cada fila (en este caso, **y**). El arreglo **z** debe tener el mismo número de columnas como el número de elementos en **x** y debe tener el mismo número de filas como el número de elementos en **y**. El cuarto y quinto campos corresponden al valor de **x** y el valor de **y** para el que le gustaría determinar nuevos valores **z**.

MATLAB también incluye una función, **interp3**, para interpolación tridimensional. Consulte la característica **help** para los detalles acerca de cómo usar esta función e **interpn**, que le permite realizar interpolación n-dimensional. Todas estas funciones tienen por defecto la técnica de interpolación lineal, pero aceptarán cualquiera de las otras técnicas mencionadas en la tabla 12.1.

Ejercicio de práctica 12.2

Cree vectores x y y para representa los siguientes datos:

$y\downarrow / x\rightarrow$	**15**	**30**
10	z = 23	33
20	45	55
30	60	70
40	82	92
50	111	121
60	140	150
70	167	177
80	198	198
90	200	210
100	20	230

1. Grafique ambos conjuntos de datos yz en la misma gráfica. Agregue una leyenda que identifique cuál valor de x aplica a cada conjunto de datos.
2. Use interpolación lineal bidimensional para aproximar el valor de z cuando $y = 15$ y $x = 20$.

3. Use interpolación cúbica segmentaria (spline) para aproximar el valor de z cuando $y = 15$ y $x = 20$.
4. Use interpolación lineal para crear una nueva subtabla para $x = 20$ y $x = 25$ para todos los valores y.

12.2 AJUSTE DE CURVAS

Idea clave: el ajuste de curva es una técnica para el modelado de datos con una ecuación.

Aunque se podrían usar técnicas de interpolación para encontrar valores de y entre valores x medidos, sería más conveniente si se pudieran modelar los datos experimentales como $y = f(x)$. Entonces se podría calcular cualquier valor de y que se quisiera. Si se sabe algo acerca de la relación subyacente entre x y y, podría ser capaz de determinar una ecuación sobre la base de dichos principios. Por ejemplo, la ley del gas ideal se basa en dos suposiciones subyacentes:

- Todas las moléculas en un gas chocan elásticamente.
- Las moléculas no ocupan espacio en su contenedor.

Ninguna de estas suposiciones es completamente precisa, de modo que la ley del gas ideal funciona sólo cuando hay una buena aproximación de la realidad, pero esto es cierto para muchas situaciones, y la ley del gas ideal es extremadamente valiosa. Sin embargo, cuando los gases reales se desvían de esta relación simple, se tienen dos opciones para cómo modelar su comportamiento: se puede intentar entender la física de la situación y ajustar la ecuación en concordancia o se puede tomar los datos y modelarlos empíricamente. Las ecuaciones empíricas no se relacionan con teoría alguna de por qué ocurre un comportamiento; sólo hacen un buen trabajo de predicción acerca de cómo cambia un parámetro en relación con otro parámetro.

MATLAB tiene funciones internas de ajuste de curvas que le permiten modelar los datos empíricamente. Es importante recordar que estos modelos son buenos sólo en la región donde se recopilaron los datos. Si no se entiende por qué un parámetro como y cambia como lo hace con x, no puede predecir si la ecuación de ajuste de datos todavía funcionará afuera del rango donde se recopilaron los datos.

12.2.1 Regresión lineal

La forma más simple de modelar un conjunto de datos es una línea recta. Vuelva a revisar los datos de la sección 12.1.1:

```
x=0:5;
y=[15, 10, 9, 6, 2, 0];
```

Si grafica los datos en la figura 12.7, puede intentar dibujar una línea recta a través de los puntos de datos para obtener un modelo burdo del comportamiento de los datos. Este proceso a veces se denomina “a ojo de buen cubero”, lo que significa que no se realizaron cálculos, pero que parece un buen ajuste.

Al observar la gráfica, puede ver que muchos de los puntos parecen caer exactamente en la línea, pero otros están afuera por cantidades variables. Para comparar la calidad del ajuste de esta línea con otros posibles estimados se debe encontrar la diferencia entre el valor y real y el valor calculado del estimado.

Se puede encontrar la ecuación de la línea en la figura 12.7 al notar que $x = 0$, $y = 15$, y en $x = 5$, $y = 0$. Por tanto, la pendiente de la línea es

$$\frac{\text{elevación}}{\text{carrera}} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{0 - 15}{5 - 0} = -3$$

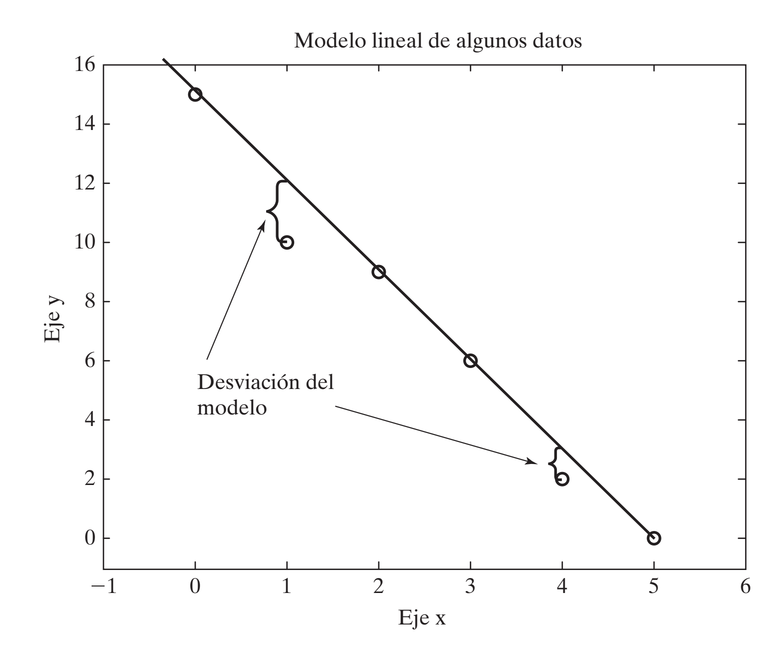


Figura 12.7
Modelo lineal; la línea fue echa "a ojo de buen cubero".

La línea cruza el eje *y* en 15, de modo que la ecuación de la línea es

$$y = -3x + 15$$

Las diferencias entre los valores reales y los valores calculados se presentan en la tabla 12.4.

regresión lineal: técnica para modelar datos como una línea recta

La técnica de *regresión lineal* usa un enfoque llamado mínimos cuadrados para comparar qué tan diferentes son las ecuaciones que modelan el comportamiento de los datos. En esta técnica, las diferencias entre los valores reales y calculados se elevan al cuadrado y se suman. Esto tiene la ventaja de que las desviaciones positivas y negativas no se cancelan mutuamente. Se podría usar MATLAB para calcular este parámetro para los datos. Se tiene

```
suma_de_cuadrados = sum((y-y_calc).^2)
```

que produce

```
suma_de_cuadrados =
  5
```

Tabla 12.4 Diferencia entre valores reales y calculados

x	y (real)	y_calc (calculado)	diferencia = y − y_calc
0	15	15	0
1	10	12	−2
2	9	9	0
3	6	6	0
4	2	3	−1
5	0	0	0

Está más allá del ámbito de este texto explicar cómo funciona la técnica de regresión lineal, excepto decir que compara diferentes modelos y elige el modelo en el que la suma de los cuadrados es la más pequeña. La regresión lineal se logra en MATLAB con la función **polyfit**. Se requieren tres campos para **polyfit**: un vector de valores *x*, un vector de valores *y* y un entero que indique qué orden de polinomio se usaría para ajustar los datos. Dado que una línea recta es un polinomio de primer orden, se ingresará el número 1 en la función **polyfit**:

```
polyfit(x,y,1)
ans =
 -2.9143 14.2857
```

Los resultados son los coeficientes correspondientes a la ecuación polinomial de primer orden de mejor ajuste:

$$y = -2.9143x + 14.2857$$

¿Realmente esto es un mejor ajuste que el modelo "de buen cubero"? Se puede calcular la suma de los cuadrados para encontrar:

```
best_y=-2.9143*x+14.2857;
new_sum=sum((y-best_y).^2)
new_sum =
  3.3714
```

Dado que el resultado del cálculo de suma de cuadrados de hecho es menor que el valor encontrado por la línea "buen cubero", se puede concluir que MATLAB encontró un mejor ajuste a los datos. Se puede graficar los datos y la línea de mejor ajuste determinada por regresión lineal (véase la figura 12.8) para intentar y obtener un sentido visual de si la línea ajusta bien los datos:

```
plot(x,y,'o',x,best_y)
```

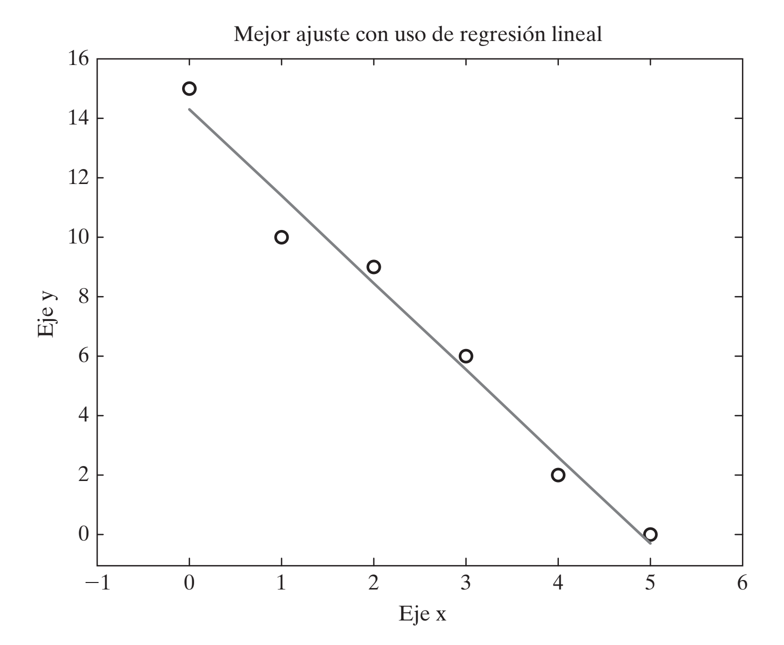


Figura 12.8
Datos y línea de mejor ajuste con el uso de regresión lineal.

12.2.2 Regresión polinomial

Desde luego, las líneas rectas no son las únicas ecuaciones que se podrían analizar con la técnica de regresión. Por ejemplo, un enfoque común es ajustar los datos con un polinomio de orden superior de la forma

$$y = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n$$

La *regresión polinomial* se usa para obtener el mejor ajuste al minimizar la suma de los cuadrados en las desviaciones de los valores calculados de los datos. La función **polyfit** le permite hacer esto fácilmente en MATLAB. Se puede ajustar los datos de muestra a ecuaciones de segundo y tercer orden con los comandos

```
polyfit(x,y,2)
ans =
  0.0536 -3.1821 14.4643
```

y

```
polyfit(x,y,3)
ans =
 -0.0648 0.5397 -4.0701 14.6587
```

que corresponden a las siguientes ecuaciones

$$y_2 = 0.0536x^2 - 3.1821x + 14.4643$$
$$y_3 = -0.0648x^3 + 0.5397x^2 - 4.0701x + 14.6587$$

Se puede encontrar la suma de los cuadrados para determinar si estos modelos ajustan mejor los datos:

```
y2=0.0536*x.^2-3.182*x + 14.4643;
sum((y2-y).^2)
ans =
  3.2643
y3=-0.0648*x.^3+0.5398*x.^2-4.0701*x + 14.6587
sum((y3-y).^2)
ans =
  2.9921
```

Como se esperaba, cuanto más términos agregue a la ecuación, "mejor" es el ajuste, al menos en el sentido de que disminuye la distancia entre los puntos de datos medidos y predichos.

Con la finalidad de graficar las curvas definidas por estas nuevas ecuaciones, necesitará más de seis puntos de datos usados en el modelo lineal. Recuerde que MATLAB crea gráficas al conectar puntos calculados con líneas rectas, así que si quiere una curva suave necesitará más puntos. Se pueden obtener más puntos y graficar las curvas con el siguiente código:

```
smooth_x=0:0.2:5;
smooth_y2=0.0536*smooth_x.^2-3.182*smooth_x + 14.4643;
subplot(1,2,1)
plot(x,y,'o',smooth_x,smooth_y2)
```

```
smooth_y3=-0.0648*smooth_x.^3+0.5398*smooth_x.^2-4.0701*
smooth_x + 14.6587;
subplot(1,2,2)
plot(x,y,'o',smooth_x,smooth_y3)
```

Idea clave: el modelado de datos se debe basar en una comprensión física del proceso, además de los datos reales recopilados.

Los resultados se muestran en la figura 12.9. Note que la ligera curvatura en cada modelo. Aunque matemáticamente estos modelos ajustan mejor los datos, pueden no ser una representación tan buena de la realidad como la línea recta. Como ingeniero o científico, necesitará evaluar cualquier modelado que haga. Necesitará considerar lo que sabe acerca de la física del proceso que modela y qué tan precisas y reproducibles son sus mediciones.

12.2.3 La función polyval

La función **polyfit** regresa los coeficientes de un polinomio que ajusta mejor los datos, al menos sobre la base de un criterio de regresión. En la sección previa se ingresaron dichos coeficientes en una expresión MATLAB para el polinomio correspondiente y se les usó para calcular nuevos valores de *y*. La función **polyval** puede realizar la misma labor sin tener que reingresar los coeficientes.

La función **polyval** requiere dos entradas. La primera es un arreglo coeficiente, como el que creó mediante **polyfit**. La segunda es un arreglo de valores *x* para el que le gustaría calcular nuevos valores *y*. Por ejemplo, se puede tener

```
coef = polyfit(x,y,1)
y_first_order_fit = polyval(coef,x)
```

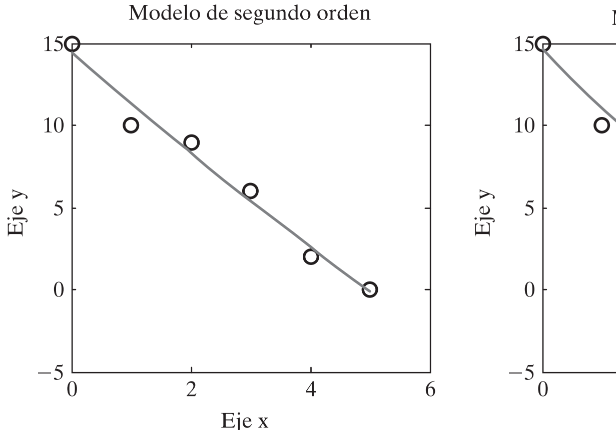
Estas dos líneas de código se podrían acortar a una línea al anidar funciones:

```
y_first_order_fit = polyval(polyfit(x,y,1),x)
```

Se puede usar la comprensión de las funciones **polyfit** y **polyval** para escribir un programa para calcular y graficar los ajustes de cuarto y quinto orden para los datos de la sección 12.1.1:

```
y4=polyval(polyfit(x,y,4),smooth_x);
y5=polyval(polyfit(x,y,5),smooth_x);

subplot(1,2,1)
plot(x,y,'o',smooth_x,y4)
axis([0,6,-5,15])
```



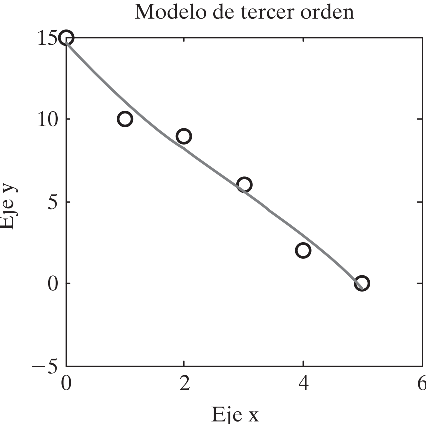
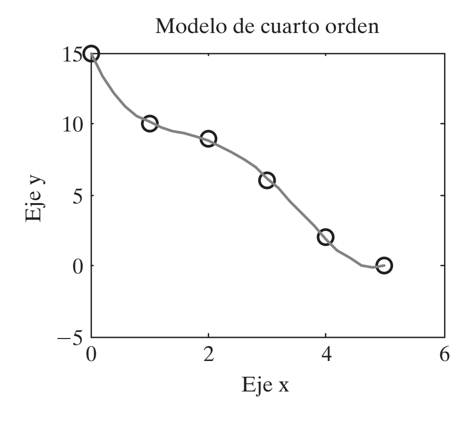


Figura 12.9
Ajustes polinomiales de segundo y tercer orden.



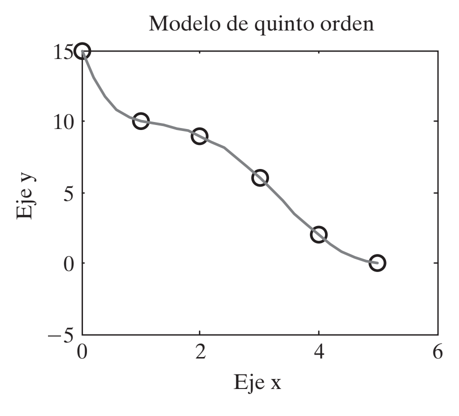


Figura 12.10
Modelos de cuarto y quinto orden de seis puntos de datos.

```
subplot(1,2,2)
plot(x,y,'o',smooth_x,y5)
axis([0,6,-5,15])
```

La figura 12.10 proporciona los resultados de la gráfica.

Como se esperaba, el ajuste de orden superior empareja los datos cada vez mejor. El modelo de quinto orden ajusta exactamente porque sólo hay seis puntos de datos.

Ejercicio de práctica 12.3

Cree vectores *x* y *y* para representar los siguientes datos:

$z = 15$		$z = 30$	
x	*y*	*x*	*y*
10	23	10	33
20	45	20	55
30	60	30	70
40	82	40	92
50	111	50	121
60	140	60	150
70	167	70	177
80	198	80	198
90	200	90	210
100	220	100	230

1. Use la función **polyfit** para ajustar los datos para $z = 15$ a un polinomio de primer orden.
2. Cree un vector de nuevos valores *x*, desde 10 hasta 100 en intervalos de 2. Use su nuevo vector en la función **polyval** junto con los valores de coeficientes encontrados en el problema 1 para crear un nuevo vector *y*.
3. Grafique los datos originales como círculos sin una línea conectora y los datos calculados como una línea sólida en la misma gráfica. ¿Qué tan bien cree que su modelo ajuste los datos?
4. Repita los problemas del 1 al 3 para los datos *x* y *y* correspondientes a $z = 30$.

EJEMPLO 12.3

Agua en un conducto

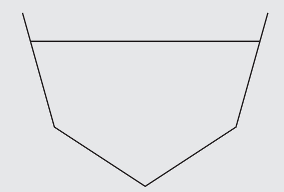


Figura 12.11
Los conductos no necesariamente tienen una sección transversal uniforme.

Determinar cuánta agua fluirá por un conducto no es tan fácil como podría parecer a primera vista. El canal podría tener una forma no uniforme (véase la figura 12.11), las obstrucciones podrían influir el flujo, la fricción es importante, etcétera. Un enfoque numérico le permite incluir todas estas preocupaciones en un modelo de cómo se comporta realmente el agua. Considere los siguientes datos recopilados de un conducto real*

Altura, ft	Flujo, ft^3/s
0	0
1.7	2.6
1.95	3.6
2.60	4.03
2.92	6.45
4.04	11.22
5.24	30.61

Calcule una ecuación lineal, cuadrática y cúbica de mejor ajuste para los datos y grafíquela en la misma gráfica. ¿Cuál modelo representa mejor los datos? (Lineal es primer orden, cuadrática es segundo orden y cúbica es tercer orden.)

1. Establezca el problema.
   Realizar una regresión polinomial sobre los datos, graficar los resultados y determinar cuál orden representa mejor los datos.
2. Describa las entradas y salidas.

   Entrada Datos de altura y flujo

   Salida Gráfica de los resultados
3. Desarrolle un ejemplo a mano.
   Dibuje una aproximación de la curva a mano. Asegúrese de comenzar en cero pues, si la altura del agua en el conducto es cero, no debería fluir agua (véase la figura 12.12).
4. Desarrolle una solución MATLAB.
   Cree la solución MATLAB en un archivo-m y luego córralo en el entorno de comandos:

```
%12.3 Ejemplo-Agua en un conducto
height = [1.7, 1.95, 2.6, 2.92, 4.04, 5.24];
flow = [2.6, 3.6, 4.03, 6.45, 11.22, 30.61];
new_height=0:0.5:6;
newf1=polyval(polyfit(height,flow,1),new_height);
newf2=polyval(polyfit(height,flow,2),new_height);
newf3=polyval(polyfit(height,flow,3),new_height);
plot(height,flow,'o',new_height,newf1,new_height,newf2,
  new_height,newf3)
title('Ajuste de flujo de agua')
xlabel('Altura de agua, ft')
ylabel('Tasa de flujo, CFS')
legend('Datos','Ajuste lineal','Ajuste cuadrático',
 'Ajuste cúbico')
```

*Tomado de Etter, Kuncicky y Moore, *Introduction to MATLAB 7* (Upper Saddle River, NJ: Pearson/Prentice Hall, 2005).

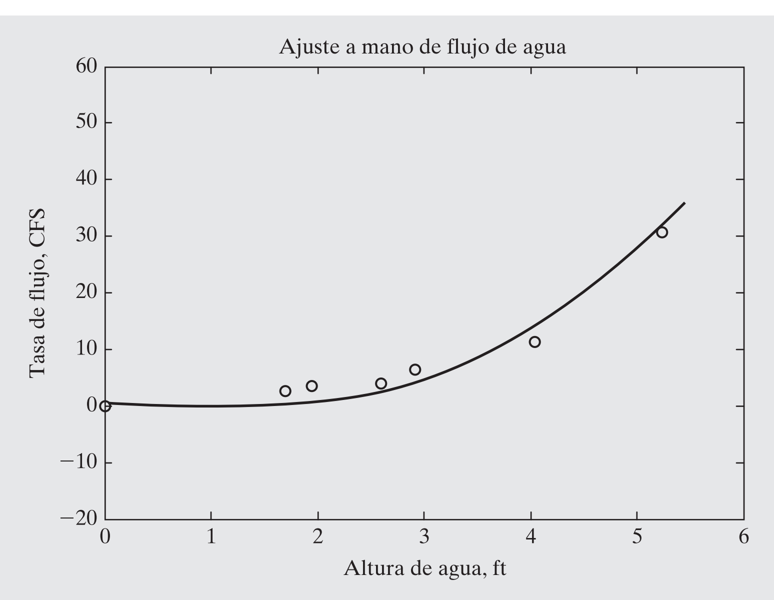


Figura 12.12
Ajuste a mano de flujo de agua.

El código MATLAB genera la gráfica que se muestra en la figura 12.13.

5. Ponga a prueba la solución.
La pregunta de cuál línea representa mejor los datos es difícil de responder. La aproximación polinomial de mayor orden seguirá mejor los puntos de datos, pero no necesariamente representa mejor la realidad.

El ajuste lineal predice que la tasa de flujo de agua será aproximadamente –5 CFS a una altura de cero, que no concuerda con la realidad. El ajuste cuadrático regresa de vuelta luego de un mínimo a una altura de aproximadamente 1.5 metros, de nuevo un resultado inconsistente con la realidad. El ajuste cúbico (tercer orden) sigue mejor los puntos y probablemente es el mejor ajuste polinomial. También se debería comparar la solución MATLAB con la solución a mano. El ajuste polinomial de tercer orden (cúbico) iguala aproximadamente la solución a mano.

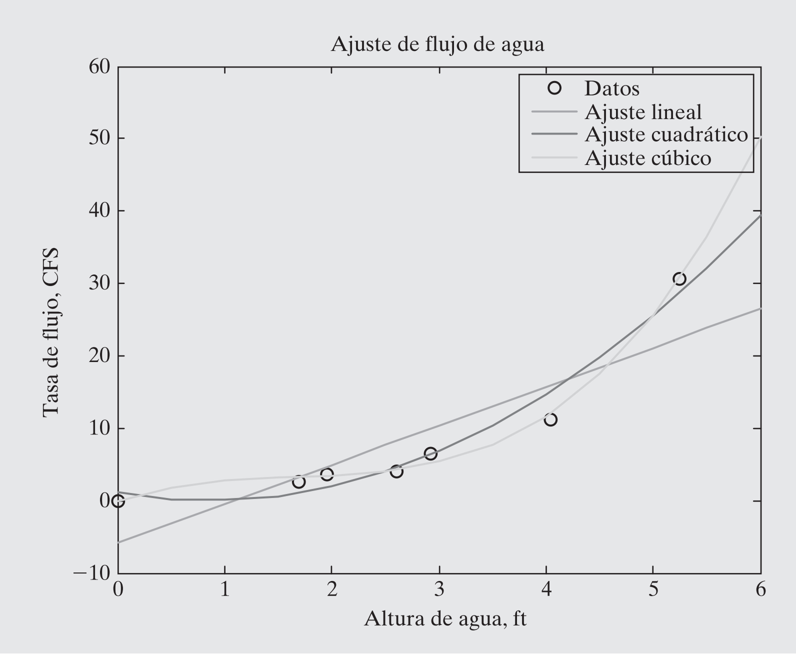


Figura 12.13
Diferentes enfoques de ajuste de curva.

EJEMPLO 12.4

Capacidad calorífica de un gas

La cantidad de energía necesaria para calentar un gas 1 grado (llamada *capacidad calorífica* del gas) depende no sólo del gas, sino también de su temperatura. Esta relación se modela usualmente con polinomios. Por ejemplo, considere los datos para dióxido de carbono en la tabla 12.5.

Use MATLAB para modelar estos datos como polinomio. Luego compare los resultados con los obtenidos del modelo publicado en B. G. Kyle, *Chemical and Process Thermodynamics* (Upper Saddle River, NJ: Prentice Hall PTR, 1999), a saber,

$$C_p = 1.698 \times 10^{-10}T^3 - 7.957 \times 10^{-7}T^2 + 1.359 \times 10^{-3}T + 5.059 \times 10^{-1}$$

1. Establezca el problema.
   Cree un modelo matemático empírico que describa la capacidad calorífica como función de la temperatura. Compare los resultados con los obtenidos de los modelos publicados.
2. Describa las entradas y salidas.

   Entrada Use la tabla proporcionada de datos de temperatura y capacidad calorífica

   Salida Encontrar los coeficientes de un polinomio que describa los datos.
   Gráfica de los resultados
3. Desarrolle un ejemplo a mano.
   Al graficar los datos (figura 12.14) se puede ver que un ajuste en línea recta (polinomio

Tabla 12.5 Capacidad calorífica de dióxido de carbono

Temperatura T en K	Capacidad calorífica C_p en kJ/(kg K)
250	0.791
300	0.846
350	0.895
400	0.939
450	0.978
500	1.014
550	1.046
600	1.075
650	1.102
700	1.126
750	1.148
800	1.169
900	1.204
1000	1.234
1500	1.328

Fuente: Tablas de propiedades térmicas de gases, NBS Circular 564, 1955.

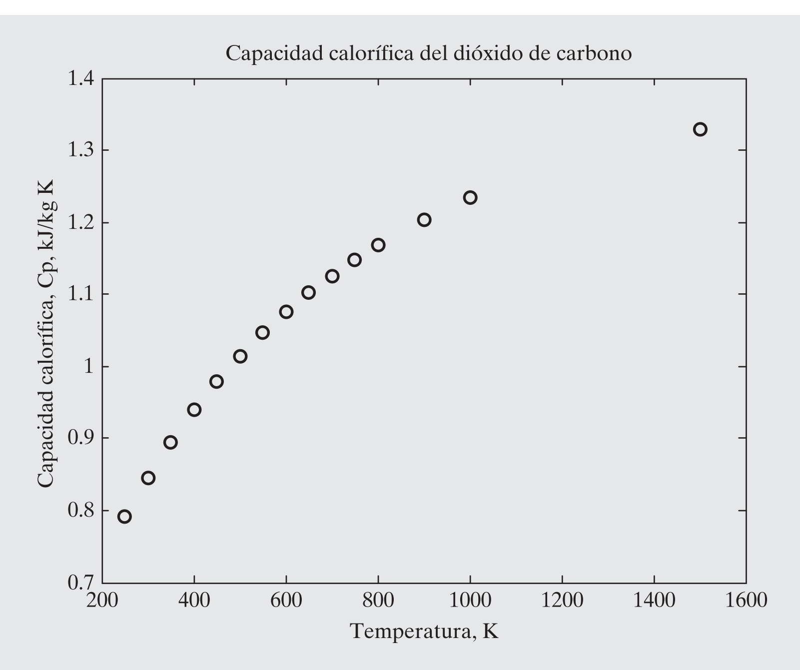


Figura 12.14
Capacidad calorífica del dióxido de carbono como función de la temperatura.

de primer orden) no es una buena aproximación de los datos. Será necesario evaluar varios modelos diferentes de, por ejemplo, primero a cuarto orden.

4. Desarrolle una solución MATLAB.

```
%Ejemplo 12.4   Capacidad calorífica de un gas

%Defina los datos medidos
T=[250:50:800,900,1000,1500];
Cp=[0.791, 0.846, 0.895, 0.939, 0.978, 1.014, 1.046, ...
1.075, 1.102, 1.126, 1.148, 1.169, 1.204, 1.234, 1.328];

%Defina un arreglo más fino de temperaturas
new_T=250:10:1500;

%Calcule nuevos valores de capacidad calorífica usando
cuatro diferentes modelos polinomiales
Cp1=polyval(polyfit(T,Cp,1),new_T);
Cp2=polyval(polyfit(T,Cp,2),new_T);
Cp3=polyval(polyfit(T,Cp,3),new_T);
Cp4=polyval(polyfit(T,Cp,4),new_T);

%Grafique los resultados
subplot(2,2,1)
plot(T,Cp,'o',new_T,Cp1)
axis([0,1700,0.6,1.6])
subplot(2,2,2)
plot(T,Cp,'o',new_T,Cp2)
axis([0,1700,0.6,1.6])
subplot(2,2,3)
plot(T,Cp,'o',new_T,Cp3)
axis([0,1700,0.6,1.6])
subplot(2,2,4)
plot(T,Cp,'o',new_T,Cp4)
axis([0,1700,0.6,1.6])
```

Si observa las gráficas que se muestran en la figura 12.15, podrá ver que un modelo de segundo o tercer orden describe adecuadamente el comportamiento en esta región de temperatura. Si decide usar un modelo polinomial de tercer orden, puede encontrar los coeficientes con **polyfit**:

```
polyfit(T,Cp,3)
ans =
2.7372e-010 -1.0631e-006 1.5521e-003 4.6837e-001
```

Los resultados corresponden a la ecuación

$$C_p = 2.7372 \times 10^{-10}T^3 - 1.0631 \times 10^{-6}T^2 + 1.5521 \times 10^{-3}T + 4.6837 \times 10^{-1}$$

5. Ponga a prueba la solución.
   Comparar los resultados con los reportados en la literatura muestra que están cerca, pero no exactos:

$$C_p = 2.737 \times 10^{-10}T^3 - 10.63 \times 10^{-7}T^2 + 1.552 \times 10^{-3}T + 4.683 \times 10^{-1}$$

(su ajuste)

$$C_p = 1.698 \times 10^{-10}T^3 - 7.957 \times 10^{-7}T^2 + 1.359 \times 10^{-3}T + 5.059 \times 10^{-1}$$

(literatura)

Esto realmente no es muy sorprendente, pues se modeló un número limitado de datos. Los modelos reportados en la literatura usan más datos y, por tanto, probablemente son más precisos.

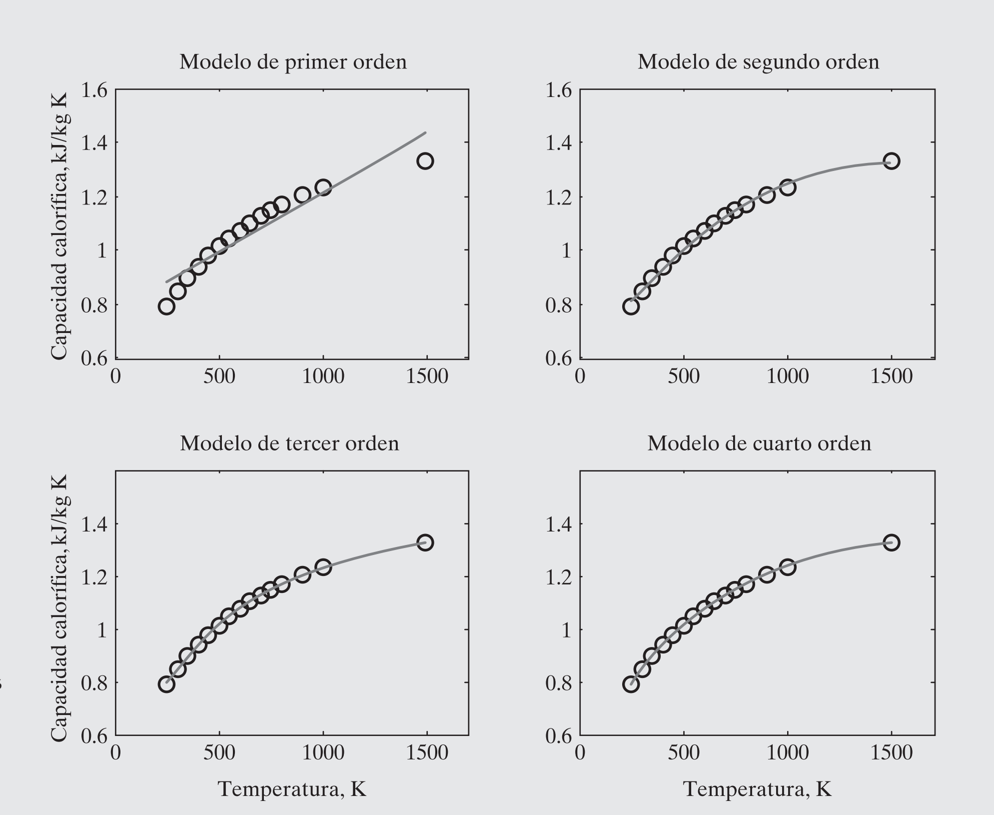


Figura 12.15
Comparación de diferentes polinomios usados para modelar los datos de capacidad calorífica de dióxido de carbono.

12.3 USO DE LAS HERRAMIENTAS DE AJUSTE INTERACTIVAS

MATLAB 7 incluye nuevas herramientas de graficación interactivas que le permiten anotar sus gráficas sin usar la ventana de comandos. También se incluyen herramientas de ajuste de curvas básico, ajuste de curvas más complicado y estadísticas.

12.3.1 Herramientas de ajuste básico

Para acceder a las herramientas de ajuste básico, primero cree una figura:

```
x=0:5;
y=[0,20,60,68,77,110]
y2=20*x;
plot(x,y,'o')
axis([-1,7,-20,120])
```

Estos comandos producen una gráfica (figura 12.16) con algunos datos muestra.

Para activar las herramientas de ajuste de curvas, seleccione **Tools → Basic Fitting** de la barra de menú en la figura. La ventana de ajuste básico se abre en la parte superior de la gráfica. Al marcar **linear**, **cubic** y **show equations** (véase la figura 12.16), se generó la gráfica que se muestra en la figura 12.17.

Al marcar el recuadro plot residuals se genera una segunda gráfica que muestra lo lejos que está cada punto de la línea calculada, como se muestra en la figura 12.18.

residual: diferencia entre el valor real y el calculado

En la esquina inferior derecha de la ventana de ajuste básico hay un botón flecha. Al seleccionar dicho botón dos veces se abre el resto de la ventana (figura 12.19).

El panel central de la ventana muestra los resultados del ajuste de curva y ofrece la opción de guardar dichos resultados en el área de trabajo. El panel derecho le permite seleccionar valores x y calcular valores y con base en la ecuación que se despliega en el panel central.

Además de la ventana de ajuste básico, puede acceder a la ventana de estadística de datos (figura 12.20) desde la barra de menú figura. Seleccione **Tools → Data Statistics** desde la

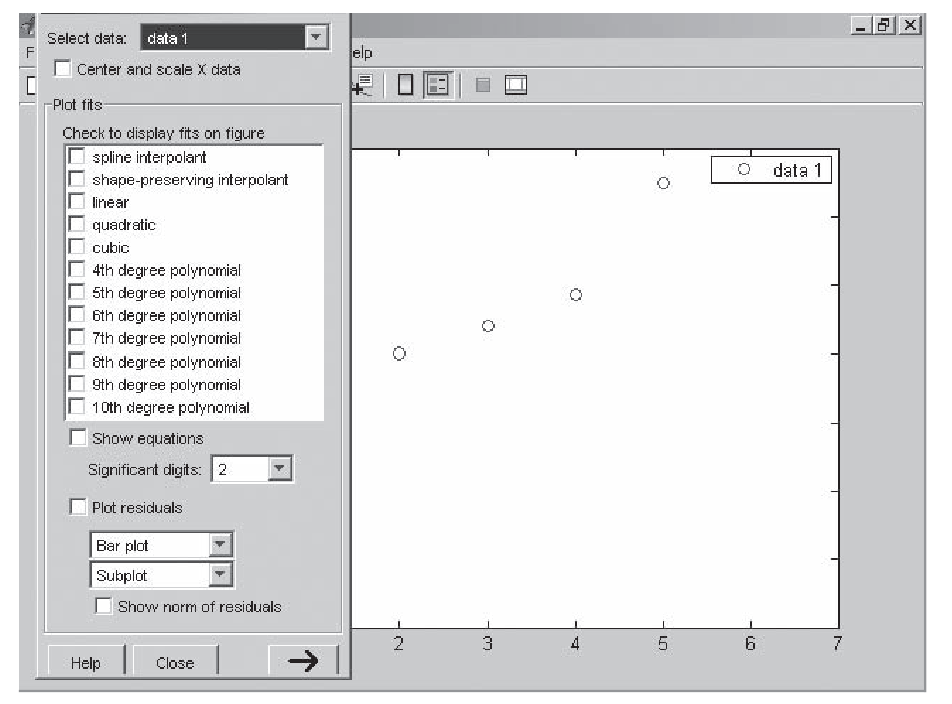


Figura 12.16
Ventana interactiva de ajuste básico.

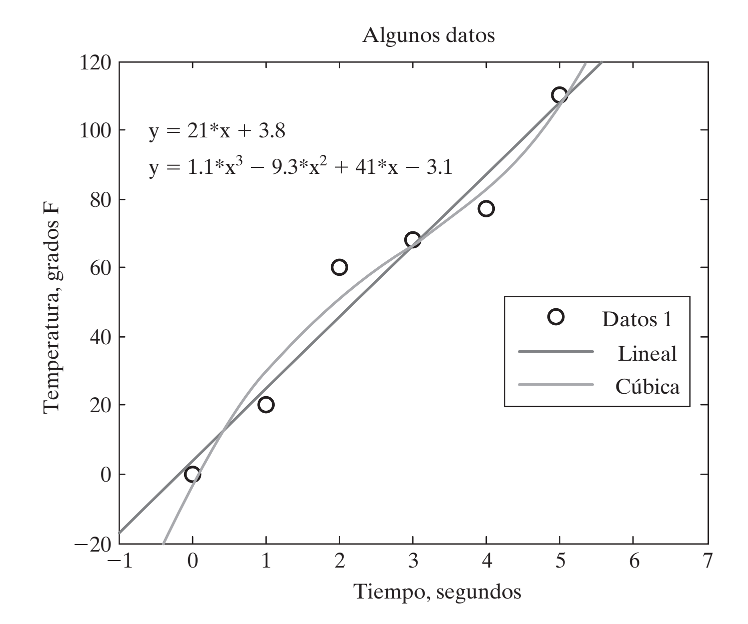


Figura 12.17
Gráfica generada con la ventana de ajuste básico.

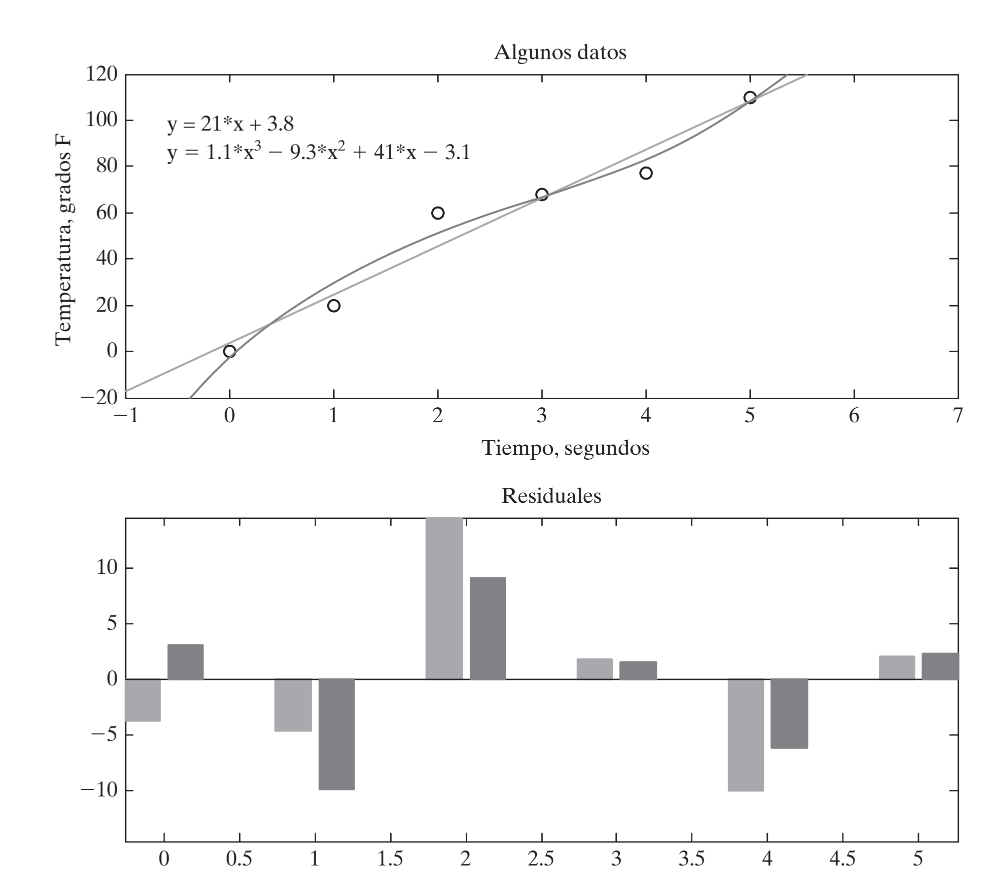


Figura 12.18
Los residuales son la diferencia entre los puntos de datos reales y calculados.

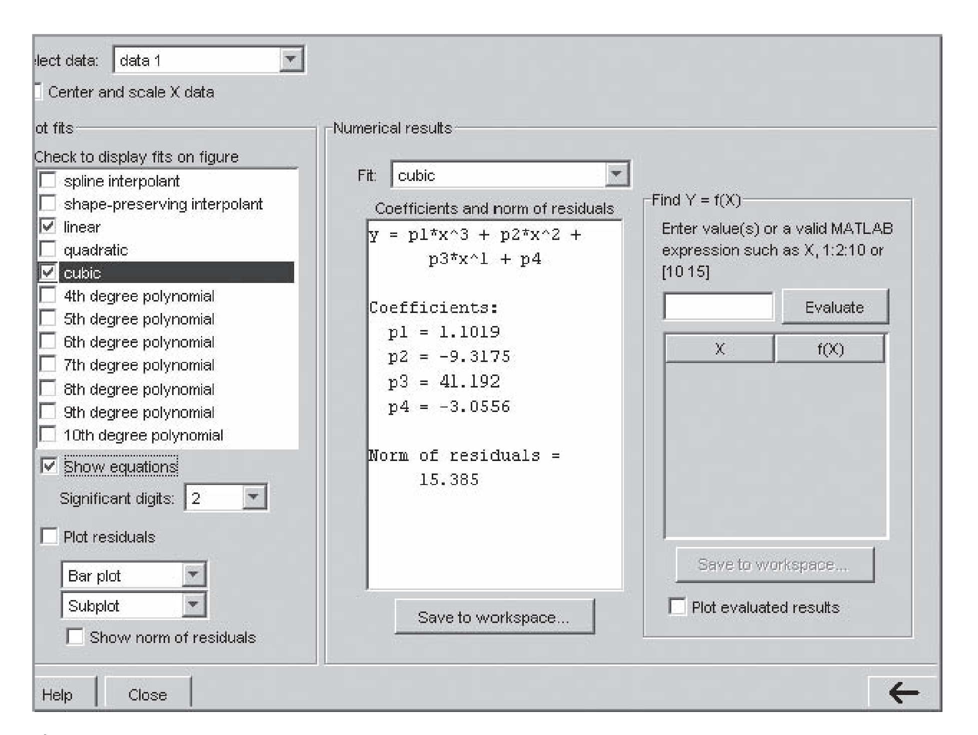


Figura 12.19
Ventana de ajuste básico.

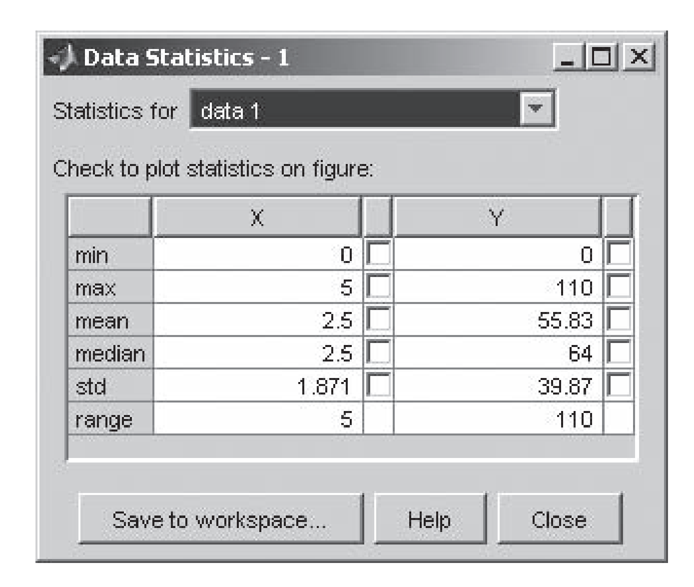


Figura 12.20
Ventana de estadísticas de datos.

ventana de figura. La ventana de estadísticas de datos le permite calcular interactivamente funciones estadísticas como la media y la desviación estándar, con base en los datos de la figura, y también le permite guardar los resultados en el área de trabajo.

12.3.2 Caja de herramientas de ajuste de curvas

Además de la utilidad de ajuste básico, MATLAB contiene cajas de herramientas para ayudarle a realizar operaciones estadísticas especializadas y de ajuste de datos. En particular, la **caja de herramientas de ajuste de curvas** contiene una interfaz gráfica de usuario (GUI) que le permite ajustar curvas con más que sólo polinomios. Sin embargo, debe tener instalada en su copia de MATLAB la caja de herramientas de ajuste de curvas, antes de que pueda ejecutar los ejemplos que siguen.

Antes de acceder a la caja de herramientas de ajuste de curvas, necesitará un conjunto de datos para analizar. Puede usar los datos que se usaron en secciones anteriores del capítulo:

```
x=0:5;
y=[0,20,60,68,77,110];
```

Para abrir la caja de herramientas de ajuste de curvas escriba

```
cftool
```

Esto lanza la ventana de herramientas de ajuste de curvas. Ahora necesitará decir a la herramienta de ajuste de curvas qué datos usar. Seleccione el botón **data**, que abrirá una ventana de datos. La ventana de datos tiene acceso al área de trabajo y le permitirá seleccionar una variable independiente (*x*) y una dependiente (*y*) de una lista desplegable. (Véase la figura 12.21.)

En el ejemplo, debe elegir **x** y **y**, respectivamente, de la lista desplegable. Puede asignar un nombre de conjunto de datos, o MATLAB le asignará un nombre por usted. Una vez que haya elegido las variables, MATLAB grafica los datos. En este punto, puede cerrar la ventana de datos.

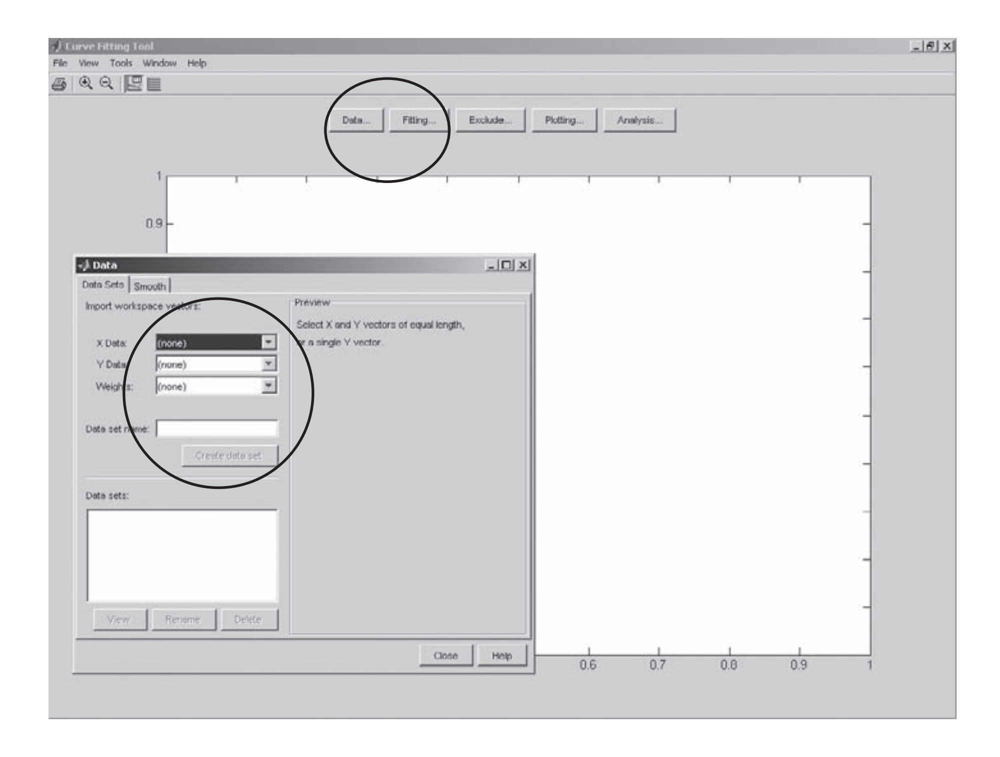


Figura 12.21
Ventanas de ajuste de curvas y datos.

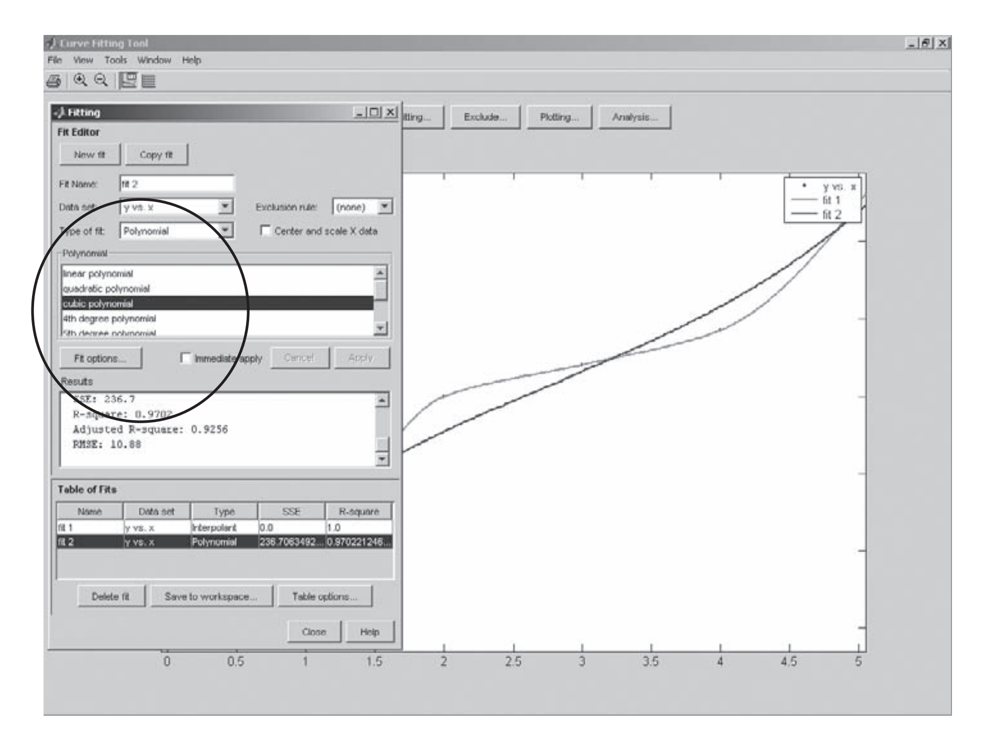


Figura 12.22
Ventana de herramientas de ajuste de curvas.

De vuelta en la ventana de herramientas de ajuste de curvas, ahora seleccione el botón **Fitting**, que le ofrece las opciones de ajustar algoritmos. Seleccione **New fit** y seleccione un tipo de ajuste de la lista **Type of fit**. Puede experimentar con opciones de ajuste para encontrar la mejor para su gráfica. En este caso se eligió un esquema interpolado, que fuerza la gráfica a través de todos los puntos, y un polinomio de tercer orden. Los resultados se muestran en la figura 12.22.

EJEMPLO 12.5

Población

La población de la Tierra se expande rápidamente (véase la figura 12.23), así como la población de Estados Unidos. MATLAB incluye un archivo de datos interno, llamado **census**, que contiene datos censales estadounidenses desde 1790. El archivo de datos contiene dos variables: **cdate**, que contiene las fechas de censo, y **pop**, que menciona la población en millones. Para cargar el archivo en su área de trabajo, escriba

```
load census
```



Figura 12.23
La población de la Tierra se expande.

Use la caja de herramientas de ajuste de curvas para encontrar una ecuación que represente los datos.

1. Establezca el problema.
   Encontrar una ecuación que represente el crecimiento poblacional de Estados Unidos.
2. Describa las entradas y salidas.

   Entrada Tabla de datos de población

   Salida Ecuación que represente los datos

3. Desarrolle un ejemplo a mano.
   Grafique los datos a mano.
4. Desarrolle una solución MATLAB.
   La caja de herramientas de ajuste de curvas es una utilidad interactiva, que se activa al escribir

```
cftool
```

lo que abre la ventana de ajuste de curvas. Debe tener instalada la caja de herramientas de ajuste de curva en su copia de MATLAB para que funcione este ejemplo. Seleccione el botón data y elija **cdate** como el valor *x* y **pop** como el valor *y*. Después de cerrar la ventana de datos, seleccione el botón fitting.

Puesto que siempre se escucha que la población crece exponencialmente, experimente con las opciones de ajuste exponencial. También se intentó la opción polinomial y se eligió un polinomio de tercer orden (cúbica). Ambos enfoques produjeron un buen ajuste, pero el polinomio en realidad fue mejor. Se envió la gráfica de la ventana de ajuste de curva a una ventana de figura y se le agregaron títulos y etiquetas (véase la figura 12.24).

A partir de los datos en la ventana de ajuste, se vio que la suma de los cuadrados de los errores (SSE) fue mayor para el ajuste exponencial, pero que ambos enfoques dieron valores *R* mayores que 0.99. (Un valor *R* de 1 indica un ajuste perfecto.)

Los resultados para el polinomio fueron los siguientes:

```
Linear model Poly3:
   f(x) = p1*x^3 + p2*x^2 + p3*x + p4
```

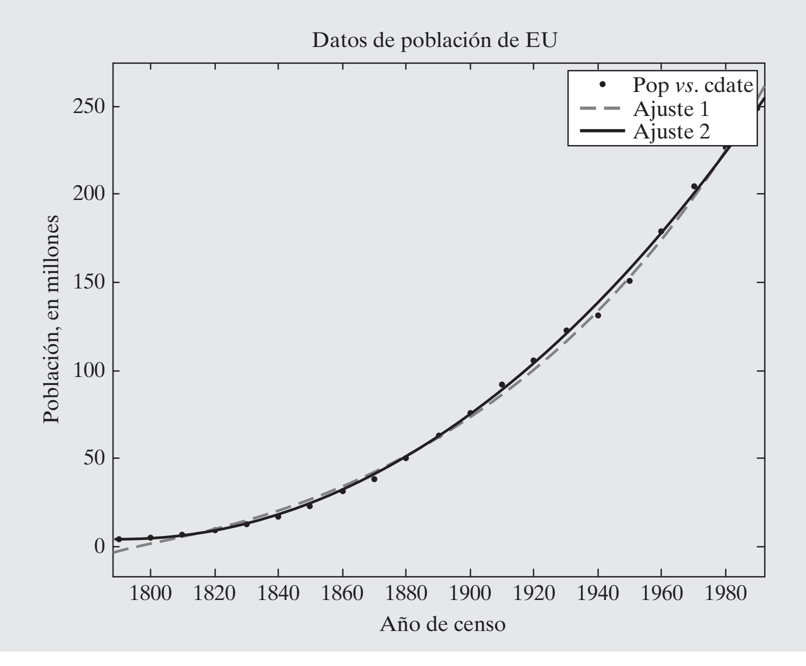


Figura 12.24
Datos censales de Estados Unidos.

```
   where x is normalized by mean 1890 and std 62.05
Coefficients (with 95% confidence bounds):
   p1 =  0.921 (-0.9743, 2.816)
   p2 =  25.18 (23.57, 26.79)
   p3 =  73.86 (70.33, 77.39)
   p4 =  61.74 (59.69, 63.8)
Goodness of fit:
 SSE: 149.8
 R-square: 0.9988
 Adjusted R-square: 0.9986
 RMSE: 2.968
```

Los valores *x* usados en la ecuación se normalizaron para un mejor ajuste al restar la media y dividir por la desviación estándar:

```
x = (cdate-mean(cdate))/std(cdate);
```

5. Ponga a prueba la solución.
   Compare los ajustes de un vistazo; ambos parecen modelar los datos adecuadamente. Es importante recordar que sólo porque una solución modele bien los datos rara vez es apropiado extender la solución a los datos medidos.

12.4 DIFERENCIAS Y DIFERENCIACIÓN NUMÉRICA

Idea clave: la función **diff** se usa tanto con expresiones simbólicas, donde encuentra la derivada, como con arreglos numéricos.

La derivada de la función $y = f(x)$ es una medida de cómo cambia *y* con *x*. Si puede definir una ecuación que relacione *x* y *y*, puede usar las funciones contenidas en la caja de herramientas simbólica para encontrar una ecuación para la derivada. Sin embargo, si todo lo que tiene son datos, puede aproximar la derivada al dividir el cambio en *y* entre el cambio en *x*:

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Si se grafican los datos de la sección 12.1, que se usaron a lo largo del capítulo, esta aproximación de la derivada corresponde a la pendiente de cada una de los segmentos de línea usados para conectar los datos, como se muestra en la figura 12.25.

Si, por ejemplo, dichos datos describen la temperatura medida de una cámara de reacción en diferentes puntos en el tiempo, las pendientes denotan la tasa de enfriamiento durante cada segmento de tiempo. MATLAB tiene una función interna llamada **diff** que encontrará la diferencia entre valores de elemento en un vector y los que se pueden usar para calcular la pendiente de pares ordenados de datos.

Por ejemplo, para encontrar el cambio en los valores **x**, escriba

```
delta_x =diff(x)
```

lo cual, dado que los valores **x** están igualmente espaciados, regresa

```
delta_x =
    1    1    1    1    1
```

De igual modo, la diferencia en los valores **y** es

```
delta_y=diff(y)
delta_y =
    -5   -1   -3   -4   -2
```

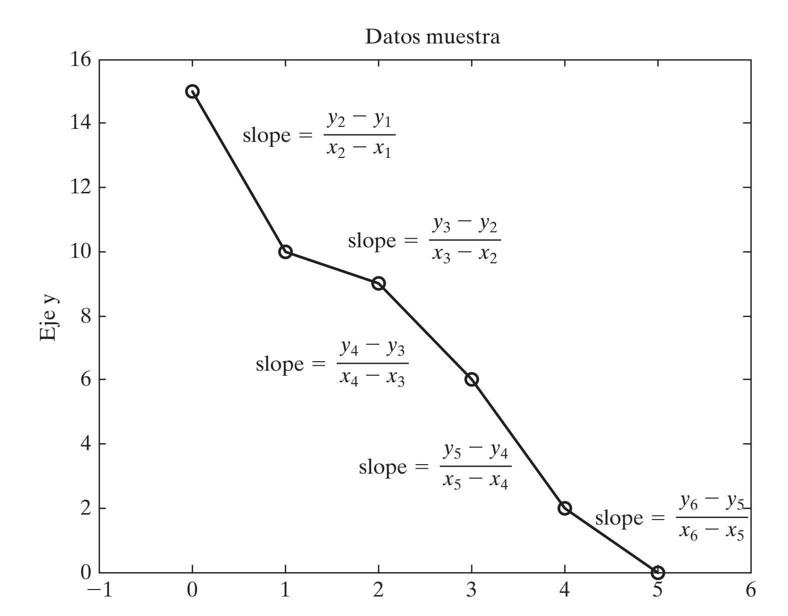


Figura 12.25
La derivada de un conjunto de datos se puede aproximar al encontrar la pendiente de una línea recta que conecta cada punto de datos.

Para encontrar la pendiente, sólo se necesita dividir **delta_y** entre **delta_x**:

```
slope=delta_y./delta_x
slope =
   -5   -1   -3   -4   -2
```

o

```
slope=diff(y)./diff(x)
slope =
   -5   -1   -3   -4   -2
```

Note que el vector que regresa cuando usa la función **diff** es un elemento más corto que el vector de entrada, porque usted calcula diferencias. Cuando usa la función **diff** para ayudarse a calcular pendientes, está calculando la pendiente entre valores de *x*, no en un valor particular. Si quiere graficar estas pendientes contra *x*, probablemente el mejor enfoque es crear una gráfica de barras, pues las tasas de cambio no son continuas. Los valores *x* se ajustaron al promedio para cada segmento de línea:

```
x=x(:,1:5)+diff(x)/2;
bar(x,slope)
```

En la figura 12.26 se muestra la gráfica de barras resultante.

La función **diff** también se puede usar para aproximar numéricamente una derivada, si se conoce la relación entre *x* y *y*. Por ejemplo, si

$$y = x^2$$

podría crear un conjunto de pares ordenados para cualquier número de valores *x*. Cuanto más valores de *x* y *y*, más suave será la gráfica. He aquí dos conjuntos de vectores *x* y *y* que se usaron para crear la gráfica de la figura 12.27a:

```
x=-2:2
y=x.^2;
```

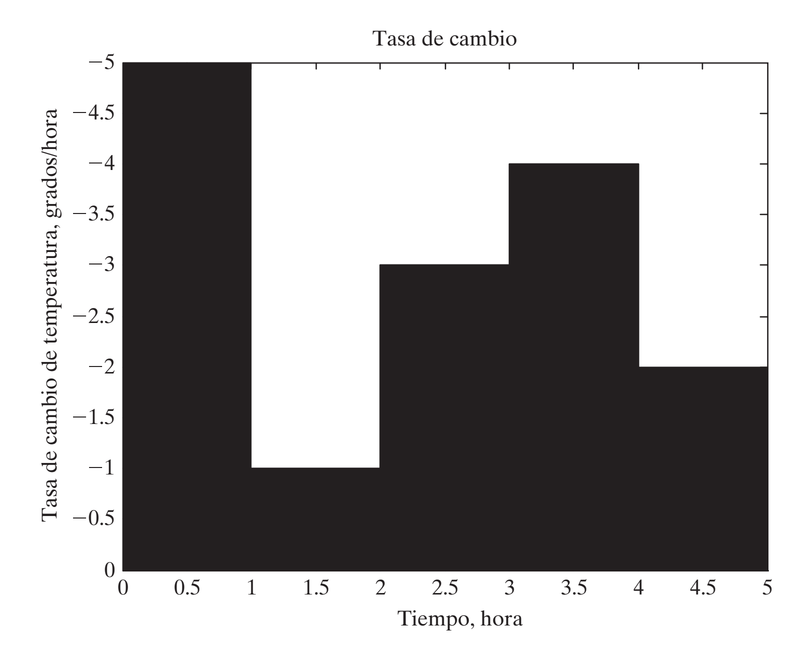


Figura 12.26
Las pendientes calculadas son discontinuas si se basan en datos. La apariencia de esta gráfica se ajustó con las herramientas de graficación interactivas.

```
big_x=-2:0.1:2;
big_y=big_x.^2;

plot(big_x,big_y,x,y,'-o')
```

Ambas líneas en la gráfica se crearon al conectar los puntos especificados con líneas rectas; sin embargo, los valores **big_x** y **big_y** están tan cerca que la gráfica parece como una curva continua. La pendiente de la gráfica *x-y* se calculó con la función **diff** y se graficó en la figura 12.27b:

```
slope5=diff(y)./diff(x);
x5=x(:,1:4)+diff(x)./2;
%Estos valores se basaron en un modelo de 5 puntos
bar(x5,slope5)
```

La gráfica de barras se modificó ligeramente con el uso de las herramientas de graficación interactivas para dar la representación que se muestra en la figura 12.27b. Se puede obtener una representación más suave, aunque todavía discontinua, con el uso de más puntos:

```
x=-2:0.5:2;
y=x.^2;

plot(big_x,big_y,x,y,'-o')

slope9=diff(y)./diff(x);
x9=x(:,1:8)+diff(x)./2;
%Estos valores se basaron en un modelo de 9 puntos
bar(x9,slope9)
```

Estos resultados se muestran en las figuras 12.27c y 12.27d. Incluso se pueden usar más puntos:

```
plot(big_x,big_y,'-o')

slope41=diff(big_y)./diff(big_x);
```

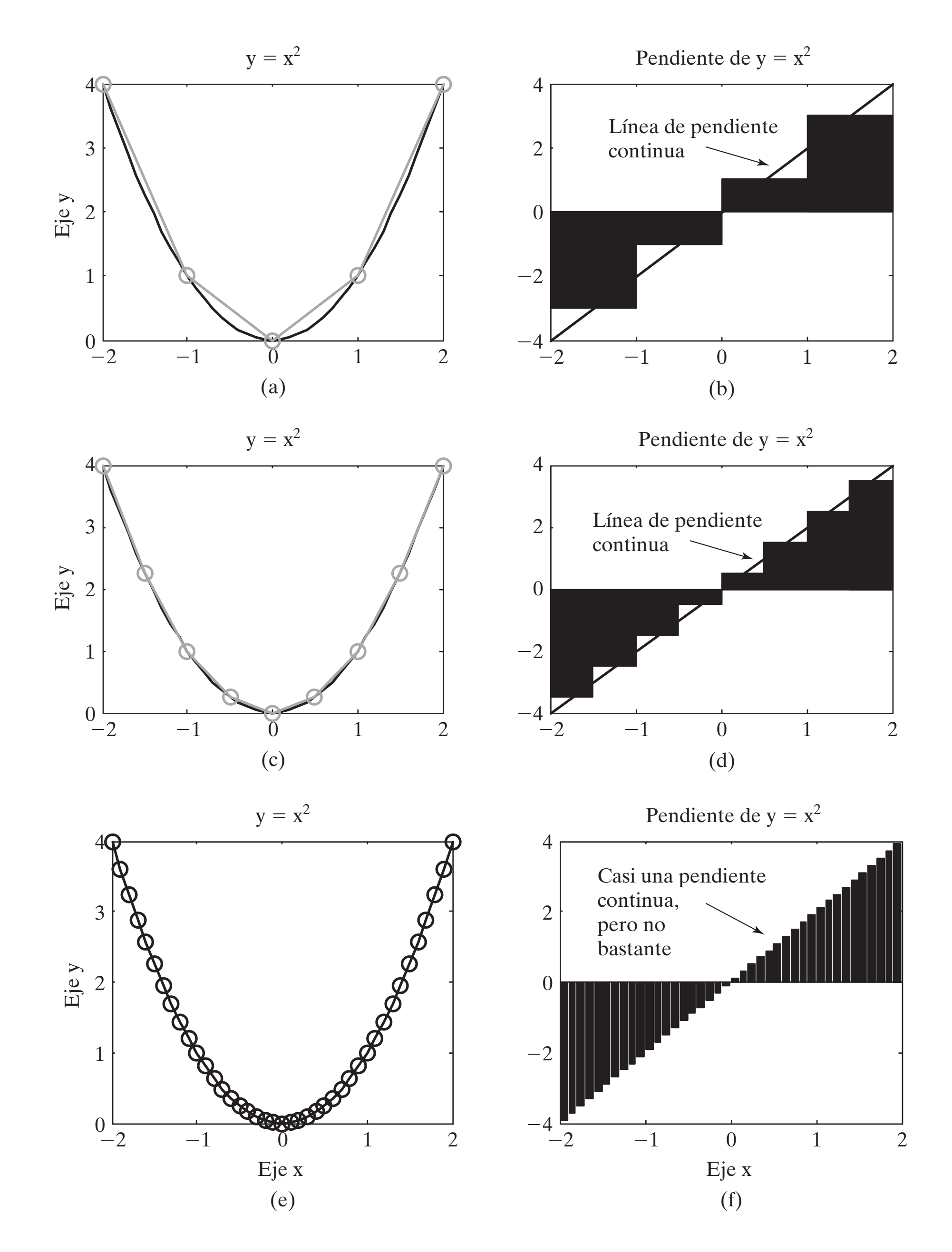


Figura 12.27
La pendiente de una función es aproximadamente más precisa cuando se usan más puntos para modelar la función.

```
x41=big_x(:,1:40)+diff(big_x)./2;%  41-point model
bar(x41,slope41)
```

Este código resulta en una representación casi suave de la pendiente como función de x, como se ve en las figuras 12.27e y 12.27f.

Ejercicio de práctica 12.4

1. Considere la siguiente ecuación:

$$y = x^3 + 2x^2 - x + 3$$

Defina un vector **x** desde –5 hasta +5 y úselo, junto con la función **diff**, para aproximar la derivada de *y* con respecto a *x*. La derivada de *y* con respecto a *x*, que se encuentra analíticamente, es

$$\frac{dy}{dx} = y' = 3x^2 + 4x - 1$$

Evalúe esta función con su vector **x** previamente definido. ¿Cómo difieren sus resultados?

2. Repita el problema 1 para las siguientes funciones y sus derivadas:

Función	Derivada
$y = \text{sen}(x)$	$\frac{dy}{dx} = \cos(x)$
$y = x^5 - 1$	$\frac{dy}{dx} = 5x^4$
$y = 5xe^x$	$\frac{dy}{dx} = 5e^x + 5xe^x$

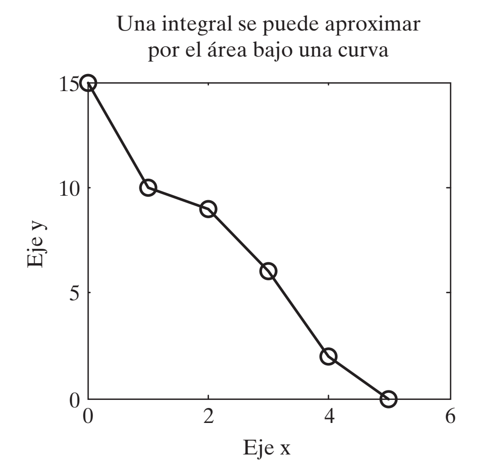
12.5 INTEGRACIÓN NUMÉRICA

Con frecuencia se considera que una integral es el área bajo una curva. Considere de nuevo los datos muestra, graficados en la figura 12.28. El área bajo la curva se puede encontrar al dividir el área en rectángulos y luego sumar las aportaciones de todos los rectángulos:

$$A = \sum_{i=1}^{n-1} (x_{i+1} - x_i)(y_{i+1} + y_i)/2$$

Los comandos MATLAB para calcular esta área son

```
avg_y=y(1:5)+diff(y)/2;
sum(diff(x).*avg_y)
```



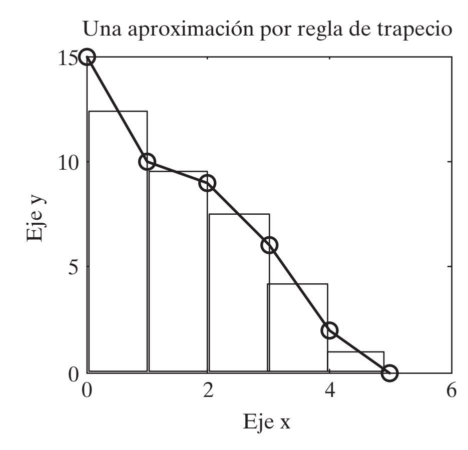


Figura 12.28
El área bajo una curva se puede aproximar con la regla del trapecio.

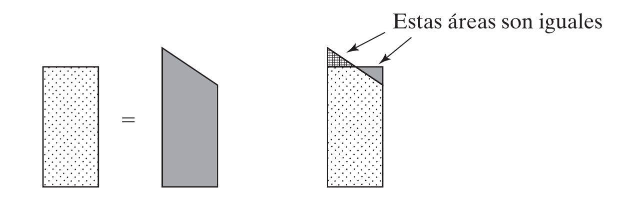


Figura 12.29
El área de un trapecio se puede modelar con un rectángulo.

A esto se le llama regla del trapecio, pues los rectángulos tienen la misma área que un trapecio dibujado entre elementos adyacentes, como se muestra en la figura 12.29.

Se puede aproximar el área bajo una curva definida por una función en lugar de datos al crear un conjunto de pares ordenados *xy*. Las mejores aproximaciones se encuentran conforme se aumenta el número de elementos en los vectores x y y. Por ejemplo, para encontrar el área bajo la función

$$y = f(x) = x^2$$

desde 0 hasta 1, se definiría un vector de valores x y calcularían los correspondientes valores y:

cuadratura: técnica para estimar el área bajo una curva con el uso de rectángulos

```
x=0:0.1:1;
y=x.^2;
```

Los valores calculados se grafican en la figura 12.30 y se usan para encontrar el área bajo la curva:

```
avg_y=y(1:10)+diff(y)/2;
sum(diff(x).*avg_y)
```

Este resultado proporciona una aproximación del área bajo la función:

```
ans =
  0.3350
```

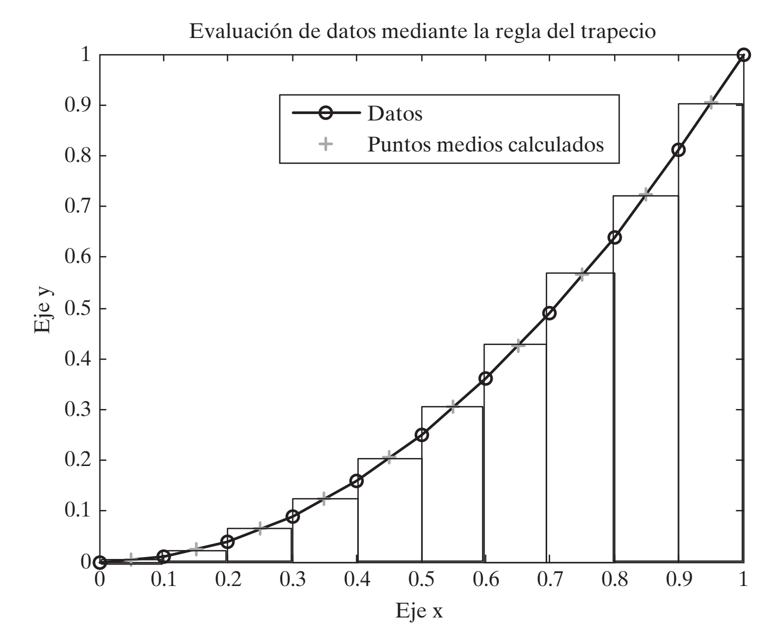


Figura 12.30
La integral de una función se puede estimar con la regla del trapecio.

La respuesta anterior corresponde a una aproximación de la integral desde $x = 0$ hasta $x = 1$, o

$$\int_0^1 x^2\,dx$$

MATLAB incluye dos funciones internas, **quad** y **quadl**, que calcularán la integral de una función sin requerir que el usuario especifique cómo se definen los rectángulos que se muestran en la figura 12.30. Las dos funciones difieren en la técnica numérica que usan. Las funciones con singularidades se pueden resolver con un enfoque u otro, dependiendo de la situación. La función **quad** usa cuadratura Simpson adaptativa:

```
quad('x.^2',0,1)
ans =
  0.3333
```

La función **quadl** usa cuadratura Lobatto adaptativa:

```
quadl('x.^2',0,1)
ans =
  0.3333
```

Sugerencia

La función **quadl** termina con la letra ‘l’, no con el número ‘1’. Puede ser difícil decir la diferencia, dependiendo de la fuente que use.

Ambas funciones requieren que el usuario ingrese una función en el primer campo. Esta función se puede llamar explícitamente como una cadena carácter, como se muestra, o se puede definir en un archivo-m o como una función anónima. Los últimos dos campos en la función definen los límites de integración, en este caso desde 0 hasta 1. Ambas técnicas se dirigen a regresar resultados dentro de un error de 1×10^{-6}. Puede encontrar más acerca de cómo funcionan estas técnicas si consulta un texto de métodos numéricos, como el de John H. Mathews y Kurtis D. Fink, *Numerical Methods Using MATLAB*, 4a ed. (Upper Saddle River, NJ: Pearson, 2004).

Ejercicio de práctica 12.5

1. Considere la siguiente ecuación:

$$y = x^3 + 2x^2 - x + 3$$

Use las funciones **quad** y **quadl** para encontrar la integral de y con respecto a x, evaluada desde –1 hasta 1. Compare sus resultados con los valores encontrados al usar la función de caja de herramientas simbólica, **int**, y la siguiente solución analítica (recuerde que las funciones **quad** y **quadl** toman entrada expresada con operadores arreglo como **.*** o **.^**, pero que la función **int** toma una representación simbólica que no usa estos operadores):

$$\int_a^b \left(x^3 + 2x^2 - x + 3\right) dx =$$

$$\left(\frac{x^4}{4} + \frac{2x^3}{3} - \frac{x^2}{2} + 3x\right)\Big|_a^b =$$

$$\frac{1}{4}\left(b^4 - a^4\right) + \frac{2}{3}\left(b^3 - a^3\right) - \frac{1}{2}\left(b^2 - a^2\right) + 3\left(b - a\right)$$

2. Repita el problema 1 para las siguientes funciones:

Función	Integral
$y = \text{sen}(x)$	$\int_a^b \text{sen}(x)dx = \cos(x)\vert_a^b = \cos(b) - \cos(a)$
$y = x^5 - 1$	$\int_a^b (x^5 - 1)dx = \left(\frac{x^6}{6} - x\right)\Big\vert_a^b = \left(\frac{b^6 - a^6}{6} - (b - a)\right)$
$y = 5x * e^x$	$\int_a^b (5e^x)dx = (-5e^x + 5xe^x)\vert_a^b =$ $(-5(e^b - e^a) + 5(be^b - ae^a))$

EJEMPLO 12.6

Cálculo de trabajo de frontera móvil

En este ejemplo se usarán las técnicas de integración numéricas de MATLAB, tanto la función **quad** como al función **quadl**, para encontrar el trabajo producido en un pistón al resolver la ecuación

$$W = \int PdV$$

con base en la suposición de que

$$PV = nRT$$

donde

P = presión, kPa,
V = volumen, m^3,
n = número de moles, kmol,
R = constante universal de gas, 8.314 kPa m^3/kmol K, y
T = temperatura, K.

También se supone que el pistón contiene 1 mol de gas a 300 K y que la temperatura permanece constante durante el proceso.

1. Establezca el problema.
   Encontrar el trabajo producido por el pistón que se muestra en la figura 12.31.

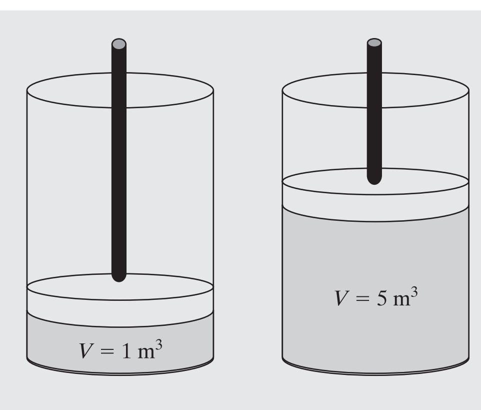


Figura 12.31
Dispositivo con pistón.

2. Describa las entradas y salidas.

   Entrada
   $T = 300$ K
   $n = 1$ kmol
   $R = 8.314$ kJ/kmol K

   $$\left.\begin{aligned} V_1 &= 1\ \text{m}^3 \\ V_2 &= 5\ \text{m}^3 \end{aligned}\right\} \quad \text{límites de la integración}$$

   Salida Trabajo realizado por el pistón

3. Desarrolle un ejemplo a mano.
   Al resolver la ley del gas ideal

   $$PV = nRT$$

   o

   $$P = nRT/V$$

   para P y realizar la integración se obtiene

   $$W = \int \frac{nRT}{V} dV = nRT \int \frac{dV}{V} = nRT \ln\left(\frac{V_2}{V_1}\right)$$

   Al sustituir los valores, se encuentra que

   $$W = 1 \text{ kmol} \times 8.314 \text{ kJ/kmol K} \times 300 \text{ K} \times \ln\left(\frac{V_2}{V_1}\right)$$

   Dado que los límites de integración son $V_2 = 5$ m^3 y $V_1 = 1$ m^3, el trabajo resultante es

   $$W = 4014 \text{ kJ}$$

   Dado que el trabajo es positivo, se produce por (y no sobre) el sistema.

4. Desarrolle una solución MATLAB.

```
%Ejemplo 12.6
%Cálculo de trabajo frontera, con funciones de cuadratura
%MATLAB
clear, clc
```

```
%Defina constantes
n=1;        % número de moles de gas
R=8.314;    % constante universal de gas
T=300;      % Temperatura, en K

%Defina una función anónima para P
P=@(V) n*R*T./V;

%Use quad para evaluar la integral
quad(P,1,5)
%Use quadl para evaluar la integral
quad(P,1,5)
```

que regresa los siguientes resultados en la ventana de comandos

```
ans =
 4.0143e+003
ans =
 4.0143e+003
```

Note que en esta solución se definió una función anónima para **P**. Se podría haber definido fácilmente la función con el uso de una cadena carácter dentro de las funciones **quad** y **quadl**. Sin embargo, en este caso se habría tenido que sustituir las variables con valores numéricos:

```
quad('1*8.314*300./V',1,5)
ans =
 4.0143e+003
```

La función también se pudo haber definido en un archivo-m.

5. Ponga a prueba la solución.
   Se comparan los resultados con la solución a mano. Los resultados son iguales. También ayudan a obtener una solución desde la caja de herramientas simbólica. ¿Por qué se necesitan ambos tipos de solución MATLAB? Porque hay algunos problemas que no se pueden resolver con las herramientas simbólicas de MATLAB y hay otros (los que tienen singularidades) que no son adecuados para un enfoque numérico.

12.6 RESOLUCIÓN NUMÉRICA DE ECUACIONES DIFERENCIALES

MATLAB incluye algunas funciones que resuelven numéricamente ecuaciones diferenciales ordinarias de la forma

$$\frac{dy}{dt} = f(t, y)$$

Las ecuaciones diferenciales de orden superior (y sistemas de ecuaciones diferenciales) se deben reformular en un sistema de ecuaciones de primer orden. (La característica **help** de MATLAB describe una estrategia para reformular su problema en esta forma.) Esta sección resalta las principales características de las funciones del solucionador de ecuaciones diferenciales ordinarias. Para más información, consulte la característica **help**.

No toda ecuación diferencial se puede resolver con la misma técnica, de modo que MATLAB incluye una gran variedad de solucionadores de ecuaciones diferenciales (tabla 12.6). Sin embargo, todos estos solucionadores tienen el mismo formato. Esto hace fácil intentar diferentes técnicas al sólo cambiar el nombre de función.

Tabla 12.6 Solucionadores de ecuaciones diferenciales de MATLAB

Función solucionador de ecuación diferencial ordinaria	Tipo de problemas probablemente resueltos con esta técnica	Método de solución numérica	Comentarios
ode45	ecuaciones diferenciales no rígidas	Runge-Kutta	Mejor elección para una técnica de primera suposición si no sabe mucho acerca de la función. Usa una fórmula explícita Runge-Kutta (4,5) llamada par Durmiente-Príncipe
ode23	ecuaciones diferenciales no rígidas	Runge-Kutta	Esta técnica usa un par explícito Runge-Kutta (2,3) de Bogacki y Shampine. Si la función es "levemente rígida", éste puede ser un mejor enfoque que **ode 45**
ode113	ecuaciones diferenciales no rígidas	Adams	A diferencia de **ode 45** y **ode 23**, que con solucionadores de un solo paso, esta técnica es un solucionador multipaso
ode15s	ecuación diferencial rígida y ecuaciones algebraica diferenciales	NDFs (BDFs)	Usa fórmulas de diferenciación numérica (NDF) o fórmulas de diferenciación hacia atrás (BDF). Es difícil predecir cuál técnica funcionará mejor en una ecuación diferencial rígida
ode23s	ecuaciones diferenciales rígidas	Rosenbrock	Formulación Rosenbock de segundo orden modificada
ode23t	ecuaciones diferenciales moderadamente rígidas y ecuaciones algebraicas diferenciales	trapezoid rule	Útil si necesita una solución sin amortiguamiento numérico
ode23tb	ecuaciones diferenciales rígidas	TR-BDF2	Este solucionador usa una fórmula implícita Runge-Kutta con la regla del trapecio (TR) y una fórmula de diferenciación hacia atrás de segundo orden (BDF2)
ode15i	ecuaciones diferenciales completamente implícitas	BDF	Este solucionador usa una fórmula de diferencia hacia atrás (BDF) para resolver ecuaciones diferenciales implícitas de la forma $f(y,y',t) = 0$

Cada uno de los solucionadores requiere las siguientes tres entradas como mínimo:

- Un manipulador de función para una función que describa la ecuación diferencial de primer orden o sistema de ecuaciones diferenciales en términos de t y y.
- El lapso de tiempo de interés.
- Una condición inicial para cada ecuación en el sistema.

Idea clave: MATLAB incluye una gran familia de solucionadores de ecuaciones diferenciales.

Todos los solucionadores regresan un arreglo de valores t y y:

```
[t,y] = odesolver(function_handle,[initial_time,
  final_time], [initial_cond_array])
```

Si no especifica los arreglos resultantes **[t,y]**, las funciones crean una gráfica de los resultados.

12.6.1 Entrada de manipulador de función

Un manipulador de función es un "apodo" para una función. El manipulador de función puede hacer referencia o a una función MATLAB estándar, almacenada como archivo-m, o a una función MATLAB anónima.

He aquí un ejemplo de una función anónima para una sola ecuación diferencial simple:

`my_fun = @(t,y) 2*t` que corresponde a $\frac{dy}{dt} = 2t$

Aunque esta función particular no usa un valor de y en el resultado ($2t$), todavía necesita ser parte de la entrada.

Si quiere especificar un sistema de ecuaciones, probablemente es más fácil definir un archivo-m de función. La salida de la función debe ser un vector columna de valores de primera derivada, como en

```
function dy=another_fun(t,y)
dy(1)= y(2);
dy(2)= -y(1);
dy=[dy(1); dy(2)];
```

Esta función representa el sistema

$$\frac{dy}{dt} = x$$

$$\frac{dx}{dt} = -y$$

que también se podría expresar en una notación más compacta como

$$y_1' = y_2$$

$$y_2' = -y_1$$

donde la prima indica la derivada con respecto al tiempo y las funciones con respecto al tiempo son y_1, y_2, etcétera. En esta notación, la segunda derivada es igual a y'' y la tercera derivada es y''':

$$y' = \frac{dy}{dt}, \quad y'' = \frac{d^2y}{dt^2}, \quad y''' = \frac{d^3y}{dt^3}$$

12.6.2 Resolución del problema

Tanto el lapso de tiempo de interés como las condiciones iniciales para cada ecuación se ingresan como vectores en los solucionadores de ecuaciones, junto con el manipulador de función. Para demostrar, resuelva la ecuación

$$\frac{dy}{dt} = 2t$$

En la sección previa se creó una función anónima para esta ecuación diferencial ordinaria y se le llamó **my_fun**. Se evaluará y desde –1 hasta 1 y la condición inicial se especifica como

$$y(-1) = 1$$

Si no sabe cómo se comporta su ecuación o sistema de ecuaciones, su primer intento debe ser **ode45**:

```
[t,y]=ode45(my_fun,[-1,1],1)
```

Este comando regresa un arreglo de **t** valores y un arreglo correspondiente de **y** valores. Puede graficar esto usted mismo o dejar que la función solver los grafique si no especifica el arreglo de salida:

```
ode45(my_fun,[-1,1],1)
```

Los resultados se muestran en la figura 12.32 y son consistentes con la solución analítica, que es

$$y = t^2$$

Note que la primera derivada de esta función es $2t$ y que $y = 1$ cuando $t = -1$.

Cuando la función de entrada o sistema de funciones se almacena en un archivo-m, la sintaxis es ligeramente diferente. El manipulador para un archivo-m existente es **@m_nombre_archivo**. Para resolver el sistema de ecuaciones descrito en **another_fun** (de la sección anterior) se usa el comando

```
ode45(@another_fun,[-1,1],[1,1])
```

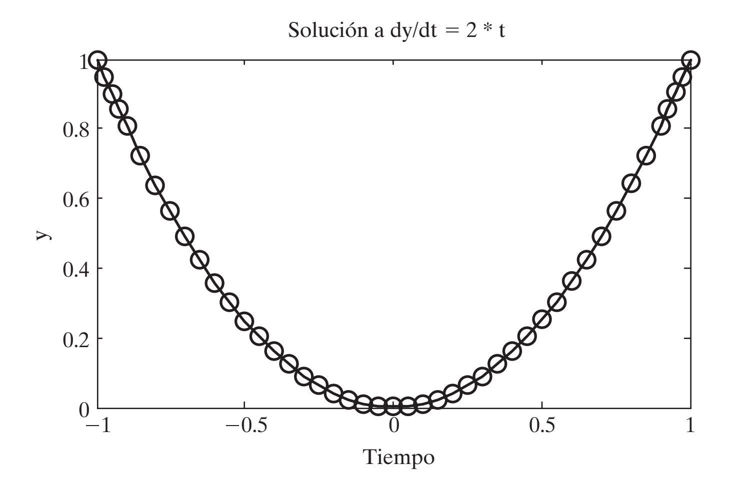


Figura 12.32
Esta figura se generó automáticamente con la función **ode45**. El título y las etiquetas se agregaron en la forma usual.

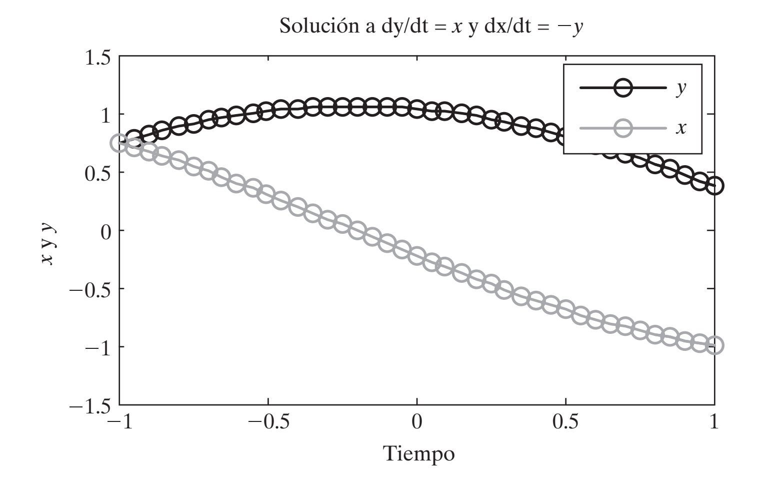


Figura 12.33
Este sistema de ecuaciones se resolvió con **ode45**. El título, etiquetas y leyenda se agregaron en la forma usual.

El lapso de tiempo de interés es desde –1 hasta 1, y las condiciones iniciales son ambas 1. Note que existe una condición inicial para cada ecuación en el sistema. Los resultados se muestran en la figura 12.33.

RESUMEN

Las tablas de datos son útiles para resumir información técnica. Sin embargo, si necesita un valor que no esté incluido en la tabla, debe aproximar dicho valor con alguna especie de técnica de interpolación. MATLAB incluye tal técnica, llamada **interp1**. Esta función requiere tres entradas: un conjunto de valores *x*, un correspondiente conjunto de valores *y* y un conjunto de valores *x* para los que le gustaría *estimar* valores *y*. La función realiza por defecto una técnica de interpolación lineal, que supone que puede aproximar dichos valores *y* intermedios como una función lineal de *x*; esto es,

$$y = f(\mathrm{x}) = ax + b$$

Para cada conjunto de dos puntos de datos se encuentra una función lineal diferente, lo que asegura que la línea que aproxima los datos siempre pase a través de los puntos tabulados.

La función **interp1** también puede modelar los datos usando aproximaciones de orden superior, de las cuales la más común es la cúbica segmentaria (spline). La técnica de aproximación se especifica como una cadena carácter en un cuarto campo opcional de la función **interp1**. Si no se especifica, la función realiza por defecto una interpolación lineal. Un ejemplo de la sintaxis es

```
new_y=interp1(tabulated_x, tabulated_y, new_x, 'spline')
```

Además de la función **interp1**, MATLAB incluye una función de interpolación bidimensional llamada **interp2**, una función de interpolación tridimensional llamada **interp3** y una función de interpolación multidimensional llamada **interpn**.

Las rutinas de ajuste de curvas son similares a las técnicas de interpolación. Sin embargo, en lugar de conectar puntos de datos, buscan una ecuación que modele los datos tan precisamente como sea posible. Una vez que tiene una ecuación, puede calcular los correspondientes valores de *y*. La curva que se modela no necesariamente pasa a través de los puntos de datos medidos. La función de ajuste de curva de MATLAB se llama **polyfit** y modela los datos como un polinomio mediante una técnica de regresión de mínimos cuadrados. La función regresa los coeficientes de la ecuación polinomial de la forma

$$y = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n$$

Estos coeficientes se pueden usar para crear la expresión apropiada en MATLAB, o se pueden usar como la entrada a la función **polyval** para calcular valores de *y* en cualquier valor de *x*. Por ejemplo, los siguientes enunciados encuentran los coeficientes de un polinomio de segundo orden que ajuste la entrada de datos *xy* y luego calculan nuevos valores de *y*, con el polinomio determinado en el primer enunciado:

```
coef = polyfit(x,y,2)
y_first_order_fit = polyval(coef,x)
```

Estas dos líneas de código se podrían recortar a una línea mediante funciones anidadas:

```
y_first_order_fit = polyval(polyfit(x,y,1),x)
```

MATLAB también incluye una capacidad interactiva de ajuste de curva que permite al usuario modelar datos no sólo con polinomios, sino con funciones matemáticas más complicadas. Se puede acceder a las herramientas de ajuste de curva básicas desde el menú **Tools** en la ventana de figura. En la caja de herramientas de ajuste de curvas están disponibles herramientas más extensivas, a las que se accede al escribir

```
cftool
```

en la ventana de comando.

Las técnicas numéricas se usan ampliamente en ingeniería para aproximar tanto derivadas como integrales. Las derivadas y las integrales también se pueden encontrar con la caja de herramientas simbólica.

La función **diff** de MATLAB encuentra la diferencia entre valores en elementos adyacentes de un vector. Al usar la función **diff** con un vector de valores *x* y un vector de valores *y*, se puede aproximar la derivada con el comando

```
slope=diff(y)./diff(x)
```

Cuanto más cercanamente espaciados estén los datos *x* y *y*, más cerca estará la aproximación de la derivada.

La integración en MATLAB se logra con una de dos funciones de cuadratura: **quad** o **quadl**. Estas funciones requieren que el usuario ingrese tanto una función como sus límites de integración. La función se puede representar como una cadena carácter tal como

```
'x.^2-1'
```

o como una función anónima, por ejemplo

```
my_function = @(x) x.^2-1
```

o como una función archivo-m, como

```
function output= my_m_file(x)
output = x.^2-1;
```

Cualquiera de las tres técnicas para definir la función se puede usar como entrada, junto con los límites de integración; por ejemplo,

```
quad('x.^2-1',1,2)
```

Tanto **quad** como **quadl** intentan regresar una respuesta precisa hasta dentro de 1×10^{-6}. Las funciones **quad** y **quadl** difieren sólo en la técnica que usan para estimar la integral. La función **quad** usa una técnica de cuadratura adaptativa Simpson, y la función **quadl** usa una técnica de cuadratura adaptativa Lobatto.

MATLAB incluye una serie de funciones solucionador para ecuaciones diferenciales ordinarias de primer orden y sistemas de ecuaciones. Todas las funciones solucionador usan el formato común

```
[t,y] = odesolver(manipulador_función,[tiempo_inicial,
  tiempo_final], [arreglo_cond_inicial])
```

Un buen primer intento usualmente es la función solucionador **ode45**, que usa una técnica Runge-Kutta. Se han formulado otras funciones solucionador para ecuaciones diferenciales rígidas y formulaciones implícitas.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona y describe brevemente todos los comandos y funciones que se definieron en este capítulo:

Comandos y funciones	
`cftool`	abre la interfaz gráfica de usuario de ajuste de curvas
`census`	conjunto de datos interno
`diff`	calcula las diferencias entre valores adyacentes en un arreglo si la entrada es un arreglo; encuentra la derivada simbólica si la entrada es una expresión simbólica
`int`	encuentra la integral simbólica
`interp1`	aproxima datos intermedios con la técnica de interpolación lineal por defecto o por un enfoque de orden superior especificado
`interp2`	función interpolación bidimensional
`interp3`	función interpolación tridimensional
`interpn`	función interpolación multidimensional
`ode45`	solucionador de ecuación diferencial ordinaria
`ode23`	solucionador de ecuación diferencial ordinaria
`ode113`	solucionador de ecuación diferencial ordinaria
`ode15s`	solucionador de ecuación diferencial ordinaria
`ode23s`	solucionador de ecuación diferencial ordinaria
`ode23t`	solucionador de ecuación diferencial ordinaria
`ode23tb`	solucionador de ecuación diferencial ordinaria
`ode15i`	solucionador de ecuación diferencial ordinaria
`polyfit`	calcula los coeficientes de un polinomio de mínimos cuadrados
`polyval`	evalúa un polinomio en un valor específico de x
`quad`	calcula la integral bajo una curva (Simpson)
`quadl`	calcula la integral bajo una curva (Lobatto)

TÉRMINOS CLAVE

aproximación
cuadratura
cuadratura Lobatto
cuadratura Simpson
cúbica segmentaria (spline)
derivada
diferenciación
ecuación cuadrática
ecuación cúbica
extrapolación
interfaz gráfica de usuario (GUI)
interpolación
interpolación lineal
mínimos cuadrados
regla trapezoidal
regresión lineal

PROBLEMAS

Interpolación

12.1 Considere un gas en un pistón en el que la temperatura se mantiene constante. Conforme cambia el volumen del dispositivo, se mide la presión. Los valores de volumen y presión se reportan en la tabla siguiente:

Volumen, m^3	Presión, kPa cuando T = 300 K
1	2494
2	1247
3	831
4	623
5	499
6	416

(a) Use interpolación lineal para estimar la presión cuando el volumen es 3.8 m^3.
(b) Use interpolación cúbica segmentaria (spline) para estimar la presión cuando el volumen es 3.8 m^3.
(c) Use interpolación lineal para estimar el volumen si la presión se mide en 1000 kPa.
(d) Use interpolación cúbica segmentaria (spline) para estimar el volumen si la presión se mide en 1000 kPa.

12.2 Con los datos del problema 12.1 e interpolación lineal cree una tabla expandida volumen-presión con mediciones de volumen cada 0.2 m^3. Grafique los valores calculados en la misma gráfica con los datos medidos. Muestre los datos medidos con círculos y sin línea, y los valores calculados con una línea sólida.
12.3 Repita el problema 12.2 con la interpolación cúbica segmentaria (spline).
12.4 El experimento descrito en el problema 12.1 se repitió a una temperatura más alta y los datos se registraron en la siguiente tabla:

Volumen, m^3	Presión, kPa a 300 K	Presión, kPa a 500 K
1	2494	4157
2	1247	2078
3	831	1386
4	623	1039
5	499	831
6	416	693

Use estos datos para resolver los siguientes ejercicios:

(a) Aproxime la presión cuando el volumen sea 5.2 m^3 para ambas temperaturas (300 y 500 K). (*Sugerencia:* haga un arreglo de presión que contenga ambos conjuntos de datos; su arreglo volumen necesitará ser 6×1 y su arreglo presión necesitará ser 6×2.) Use interpolación lineal para sus cálculos.
(b) Repita sus cálculos con interpolación cúbica segmentaria (spline).

12.5 Use los datos del problema 12.4 para resolver los siguientes problemas:

(a) Cree una nueva columna de valores presión a T = 400 K con interpolación lineal.
(b) Cree una tabla expandida volumen-presión con mediciones de volumen cada 0.2 m^3, con columnas correspondientes a T = 300 K, T = 400 K y T = 500 K.

12.6 Use la función **interp2** y los datos del problema 12.4 para aproximar un valor de presión cuando el volumen sea 5.2 m^3 y la temperatura 425 K.

Ajuste de curvas

12.7 Ajuste los datos del problema 12.1 con polinomios de primero, segundo, tercero y cuarto orden, con la función **polyfit**:

- Grafique sus resultados en la misma gráfica.
- Grafique los datos reales como círculo sin línea.

- Calcule los valores para graficar a partir de sus resultados de regresión polinomial a intervalos de 0.2 m^3.
- No muestre los valores calculados en la gráfica, pero conecte los puntos con líneas sólidas.
- ¿Cuál modelo parece hacer el mejor trabajo?

12.8 La relación entre presión y volumen usualmente no se modela mediante un polinomio. Más bien se relacionan inversamente uno con otro mediante la ley del gas ideal,

$$P = \frac{nRT}{V}$$

Esta relación se puede graficar como una línea recta si se grafica P en el eje y y $1/V$ en el eje x. Entonces la pendiente se convierte en los valores de nRT. Se puede usar la función **polyfit** para encontrar esta pendiente si se ingresa P y $1/V$ a la función:

```
polyfit(1./V, P,1)
```

(a) Si supone que el valor de n es 1 mol y el valor de R es 8.314 kPa/kmol K, muestre que la temperatura que se usa en el experimento es de hecho 300 K.
(b) Cree una gráfica con $1/V$ en el eje x y P en el eje y.

12.9 Resistencia y corriente son inversamente proporcionales una a otra en circuitos eléctricos:

$$I = \frac{V}{R}$$

Considere los siguientes datos recopilados de un circuito eléctrico al que se aplicó un voltaje constante desconocido (figura P12.9):

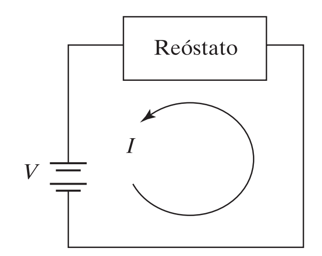


Figura P12.9
Circuito eléctrico.

Resistencia, ohms	Corriente medida, amps
10	11.11
15	8.04
25	6.03
40	2.77
65	1.97
100	1.51

(a) Grafique resistencia (R) en el eje x y corriente medida (I) en el eje y.
(b) Cree otra gráfica con $1/R$ en el eje x e I en el eje y.
(c) Use **polyfit** para calcular los coeficientes de la línea recta que se muestra en su gráfica en la parte (b). La pendiente de su línea corresponde al voltaje aplicado.
(d) Use **polyval** para encontrar valores calculados de corriente (I) con base en los resistores usados. Grafique sus resultados en una nueva figura, junto con los datos medidos.

12.10 Muchos procesos físicos se pueden modelar mediante una ecuación exponencial. Por ejemplo, las tasas de reacción química dependen de una constante de tasa de reacción que es función de la temperatura y la energía de activación:

$$k = k_0 e^{-Q/RT}$$

En esta ecuación,

R = constante universal de gas, 8.314 kJ/kmol K,
Q = energía de activación, en kJ/kmol,

T = temperatura, en K, y
k_0 = constante cuyas unidades dependen de características de la reacción. Una posibilidad es s^{-1}.

Un enfoque para encontrar los valores de k_0 y Q a partir de datos experimentales es graficar el logaritmo natural de k en el eje y y $1/T$ en el eje x. Esto debe resultar en una línea recta con pendiente $-Q/R$ e intercepta $\ln(k_0)$; esto es,

$$\ln(k) = \ln(k_0) - \frac{Q}{R}\left(\frac{1}{T}\right)$$

pues la ecuación ahora tiene la forma

$$y = ax + b$$

con $y = \ln(k)$, $x = 1/T$, $a = -Q/R$ y $b = \ln(k)$.

Ahora considere los siguientes datos:

***T*, K**	***k*, s^{-1}**
200	1.46×10^{-7}
400	0.0012
600	0.0244
800	0.1099
1000	0.2710

(a) Grafique los datos con $1/T$ en el eje x y $\ln(k)$ en el eje y.
(b) Use la función **polyfit** para encontrar la pendiente de su gráfica, $-Q/R$, y la intersección, $\ln(k_0)$.
(c) Calcule el valor de Q.
(d) Calcule el valor de k_0.

12.11 La potencia eléctrica con frecuencia se modela como

$$P = I^2R$$

donde
P = potencia, en watts,
I = corriente, en amps, y
R = resistencia, en ohms.

(a) Considere los siguientes datos y encuentre el valor del resistor en el circuito al modelar los datos como un polinomio de segundo orden con la función **polyfit**:

Potencia, W	**Corriente, amps**
50,000	100
200,000	200
450,000	300
800,000	400
1,250,000	500

(b) Grafique los datos y use las herramientas de ajuste de curvas que se encuentran en la ventana de figura para determinar el valor de R al modelar los datos como un polinomio de segundo orden.

12.12 Usar un polinomio para modelar una función puede ser muy útil, pero siempre es peligroso extrapolar más allá de sus datos. Se puede demostrar este equívoco al modelar una onda seno como un polinomio de tercer orden.

(a) Defina **x=-1:0.1:1;**.
(b) Calcule **y=sen(x)**.
(c) Use la función **polyfit** para determinar los coeficientes de un polinomio de tercer orden para modelar estos datos.
(d) Use la función **polyval** para calcular nuevos valores de **y** (**modeled_y**) con base en su polinomio, para su vector **x** desde –1 hasta 1.
(e) Grafique ambos conjuntos de valores en la misma gráfica. ¿Qué tan bien ajustan?
(f) Cree un nuevo vector **x**, **new_x=-4:0.1:4;**.
(g) Calcule valores **new_y** al encontrar **sin(new_x)**.
(h) Extrapole valores **new_modeled_y** con **polyfit**, el vector coeficiente que encontró en la parte (c) para modelar **x** y **y** entre –1 y 1, y los valores **new_y**.
(i) Grafique los dos nuevos conjuntos de valores en la misma gráfica. ¿Qué tan bueno es el ajuste afuera de la región de –1 a 1?

Aproximación de derivadas

12.13 Considere la siguiente ecuación:

$$y = 12x^3 - 5x^2 + 3$$

(a) Defina un vector **x** desde –5 hasta +5 y úselo junto con la función **diff** para aproximar la derivada de y con respecto a x.
(b) Analíticamente, se encuentra que la derivada de y con respecto a x es

$$\frac{dy}{dx} = y' = 36x^2 - 10x$$

Evalúe esta función con su vector **x** anteriormente definido. ¿Cómo difieren sus resultados?

12.14 Un uso muy común de las derivadas es para determinar velocidades. Considere los datos siguientes, tomados durante un viaje en automóvil desde Salt Lake City hasta Denver:

Tiempo, horas	Distancia millas
0	0
1	60
2	110
3	170
4	220
5	270
6	330
7	390
8	460

(a) Encuentre la velocidad promedio en mph durante cada hora del viaje.
(b) Grafique estas velocidades en una gráfica de barras. Edite la gráfica de modo que cada barra cubra 100% de la distancia entre entradas.

12.15 Considere los siguientes datos, tomados durante un viaje en automóvil desde Salt Lake City hasta Los Ángeles:

Tiempo, horas	Distancia, millas
0	0
1.0	75
2.2	145
2.9	225
4.0	300
5.2	380
6.0	430
6.9	510
8.0	580
8.7	635
9.7	700
10	720

(a) Encuentre la velocidad promedio en mph durante cada segmento del viaje.
(b) Grafique estas velocidades contra el tiempo inicial para cada segmento.
(c) Use el comando **find** para determinar si algunas de las velocidades promedio exceden el límite de rapidez de 75 mph.
(d) ¿El promedio global está por arriba del límite de rapidez?

12.16 Considere los siguientes datos del lanzamiento de un cohete de tres etapas:

Tiempo, segundos	Altitud, metros
0	0
1.00	107.37
2.00	210.00
3.00	307.63
4.00	400.00
5.00	484.60
6.00	550.00
7.00	583.97
8.00	580.00
9.00	549.53
10.00	570.00
11.00	699.18
12.00	850.00
13.00	927.51
14.00	950.00
15.00	954.51
16.00	940.00
17.00	910.68
18.00	930.00
19.00	1041.52
20.00	1150.00
21.00	1158.24
22.00	1100.00
23.00	1041.76
24.00	1050.00

(a) Cree una gráfica con tiempo en el eje x y altitud en el eje y.
(b) Use la función **diff** para determinar la velocidad durante cada intervalo de tiempo y grafique la velocidad contra el tiempo de partida para cada intervalo.
(c) Use de nuevo la función **diff** para determinar la aceleración para cada intervalo de tiempo y grafique la aceleración contra el tiempo de partida para cada intervalo.
(d) Estime los tiempos de las etapas (el tiempo cuando una etapa quemada se descarta y la siguiente etapa se enciende) al examinar las gráficas que creó.

Integración numérica

12.17 Considere la siguiente ecuación

$$y = 5x^3 - 2x^2 + 3$$

Use las funciones **quad** y **quadl** para encontrar la integral con respecto a x, evaluada de –1 a 1. Compare sus resultados con los valores que encontró con el uso de la función de la caja de herramientas simbólicas, **int**, y la siguiente solución analítica (recuerde que las funciones **quad** y **quadl** toman entrada expresada con operadores arreglo como **.*** y **.^**, pero que la función **int** toma una representación simbólica que no usa estos operadores):

$$\int_a^b (5x^3 - 2x^2 + 3)\,dx =$$
$$\left(\frac{5x^4}{4} - \frac{2x^3}{3} + 3x\right)\Bigg|_a^b =$$
$$\frac{5}{4}(b^4 - a^4) - \frac{2}{3}(b^3 - a^3) + 3(b - a)$$

12.18 La ecuación

$$C_P = a + bT + cT^2 + dT^3$$

es un polinomio empírico que describe el comportamiento de la capacidad calorífica C_p como función de la temperatura en grados K. El cambio en entalpía (una medida de energía) conforme un gas se caliente desde T_1 hasta T_2 es la integral de esta ecuación con respecto a T:

$$\Delta h = \int_{T_1}^{T_2} C_P\,dT$$

Encuentre el cambio en entalpía del oxígeno gaseoso conforme se calienta desde 300 K hasta 1000 K, con las funciones cuadratura de MATLAB. Los valores de a, b, c y d para el oxígeno son los siguientes:

$$a = 25.48$$
$$b = 1.520 \times 10^{-2}$$
$$c = -0.7155 \times 10^{-5}$$
$$d = 1.312 \times 10^{-9}$$

12.19 En algunos problemas de muestra de este capítulo, se exploraron las ecuaciones que describen trabajo de frontera móvil producido por un pistón. Una ecuación similar describe el trabajo producido conforme un gas o un líquido fluyen a través de una bomba, turbina o compresor (figura P12.19). En este caso, no hay frontera móvil, sino que hay trabajo de eje, dado por

$$\dot{W}_{\text{producido}} = -\int_{\text{entrada}}^{\text{salida}} \dot{V}\, dP$$

Esta ecuación se puede integrar si se puede encontrar una relación entre $\dot{V}$ y P. Para gases ideales, dicha relación es

$$\dot{V} = \frac{\dot{n}RT}{P}$$

Si el proceso es isotérmico, la ecuación para trabajo se convierte en

$$\dot{W} = -\dot{n}RT \int_{\text{entrada}}^{\text{salida}} \frac{dP}{P}$$

donde

$\dot{n}$ = tasa de flujo molar, en kmol/s,
R = constante universal de gas, 8.314 kJ/kmol K,
T = temperatura, en K,
P = presión, en kPa, y
$\dot{W}$ = potencia, en kW.

Encuentre la potencia producida en una turbina de gas isotérmica si

$\dot{n}$ = 0.1 kmol/s,
R = constante universal de gas, 8.314 kJ/kmol K,
T = 400 K,
P_{entrada} = 500 kPa, y
P_{salida} = 100 kPa.

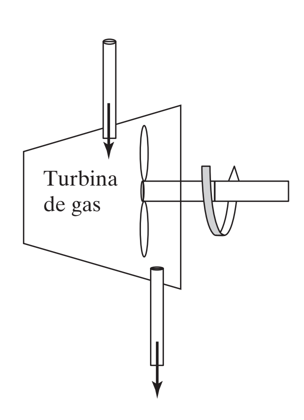


Figura P12.19

Turbina de gas usada para producir potencia.



CAPÍTULO

13

Gráficos avanzados

Objetivos

Después de leer este capítulo, el alumno será capaz de

- comprender cómo MATLAB manipula los tres diferentes tipos de archivos de imagen.
- asignar un manipulador a gráficas y ajustar propiedades, con gráficas handle.
- crear una animación con cualquiera de las dos técnicas MATLAB.
- ajustar parámetros de iluminación, ubicaciones de cámara y valores de transparencia.
- usar técnicas de visualización tanto para información escalar como vectorial en tres dimensiones.

INTRODUCCIÓN

Algunas de las gráficas usadas comúnmente en ingeniería son el caballo de batalla de la gráfica *x-y*, las gráficas polares y las gráficas de superficie, así como algunas técnicas de graficación usadas con más frecuencia en aplicaciones de negocios, como las gráficas de pastel, las gráficas de barras y los histogramas. MATLAB permite al usuario un significativo control sobre la apariencia de estas gráficas, así como manipular imágenes (como fotografías digitales) y crear representaciones tridimensionales (además de gráficas de superficie) tanto de datos como de modelos de procesos físicos.

13.1 IMÁGENES

La exploración de algunas de las capacidades gráficas más avanzadas de MATLAB comienza con el examen de cómo se manipulan las imágenes con las funciones **image** e **imagesc**. Puesto que MATLAB ya es un programa de manipulación de matrices, tiene sentido que las imágenes se almacenen como matrices.

Se puede crear una gráfica de superficie tridimensional de la función **peaks** al escribir

```
surf(peaks)
```

Se puede manipular la figura que se creó (figura 13.1) al usar las herramientas interactivas de manipulación de figura, de modo que se vea hacia abajo desde la parte superior (figura 13.2).

Una forma más sencilla de lograr el mismo objetivo es usar la gráfica de seudocolor:

```
pcolor(peaks)
```

También se pueden remover las líneas de retícula, que se grafican automáticamente, al especificar la opción shading (sombreado):

```
shading flat
```

Los colores de las figuras de la 13.1 a la 13.3 corresponden a los valores de *z*. Los valores positivos grandes de *z* son rojos (si observa los resultados en la pantalla y no en este libro que, desde luego, es negro y blanco), y los valores negativos grandes

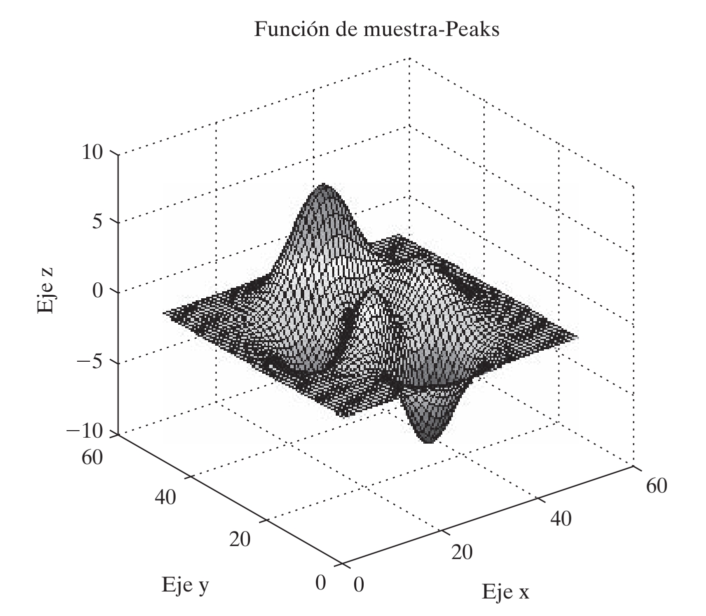


Figura 13.1
La función **peaks** se construye en MATLAB para usar en demostración de capacidades gráficas. Se agregaron el título y las etiquetas de eje.

son azules. El valor de *z* que se encuentra en el primer elemento matricial, *z*(1,1), se representa en la esquina inferior izquierda de la gráfica. (Véase la figura 13.3, derecha.)

Aunque esta estrategia para representar datos tiene sentido debido al sistema coordenado que se usa comúnmente en graficación, no tiene sentido para representar imágenes como las fotografías. Cuando las imágenes se almacenan en matrices, usualmente se representan los datos con inicio en la esquina superior izquierda de la imagen y se trabaja a través y hacia abajo (figura 13.4, izquierda). En MATLAB, existen dos funciones que se usan para desplegar imágenes: **image** e **imagesc**, que usan este formato. La función imagen escalada (**imagesc**) usa todo el mapa de colores para representar los datos, tal como la función gráfica seudocolor (**pcolor**). Los resultados que se obtienen con

```
imagesc(peaks)
```

se muestran a la derecha en la figura 13.4.

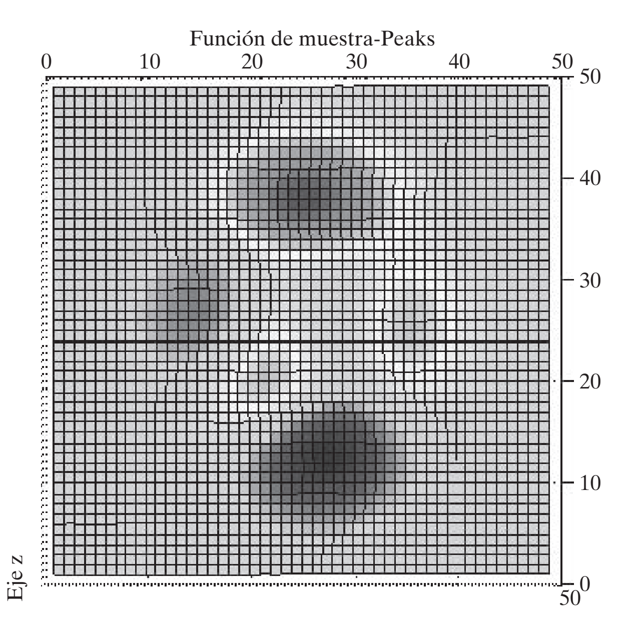
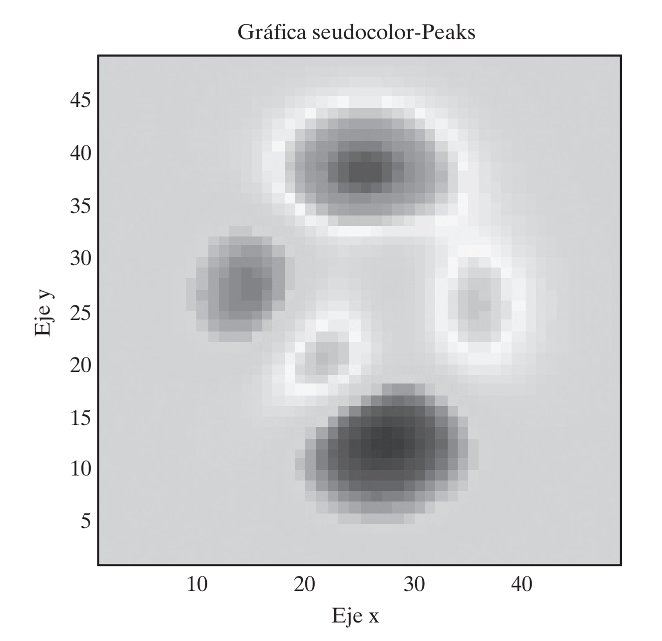


Figura 13.2
Una vista de la gráfica de superficie la función **peaks** viendo hacia abajo por el eje *z*.



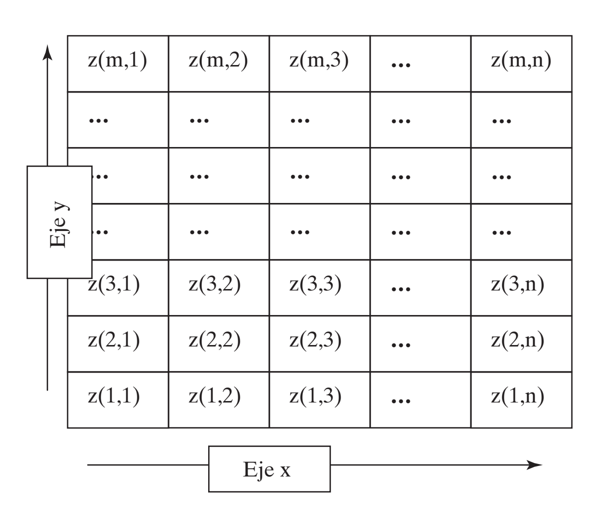
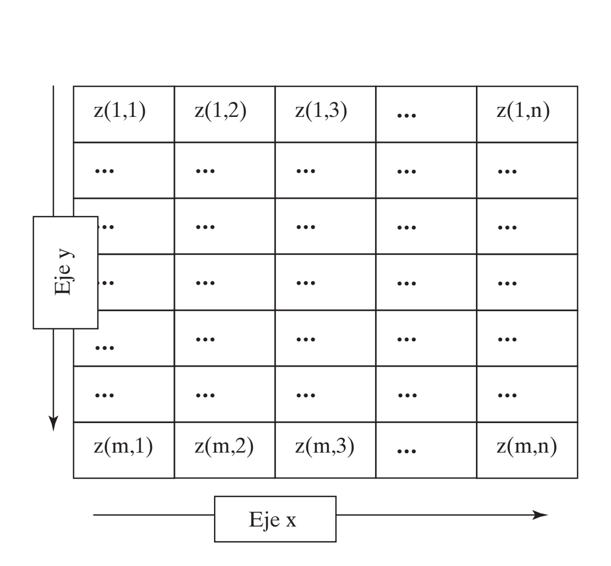


Figura 13.3
Una gráfica en seudocolor (izquierda) es lo mismo que ver justo hacia abajo una gráfica de superficie. Las gráficas de seudocolor organizan los datos sobre la base de la regla de la mano derecha, partiendo en la posición (0,0) en la gráfica (derecha).



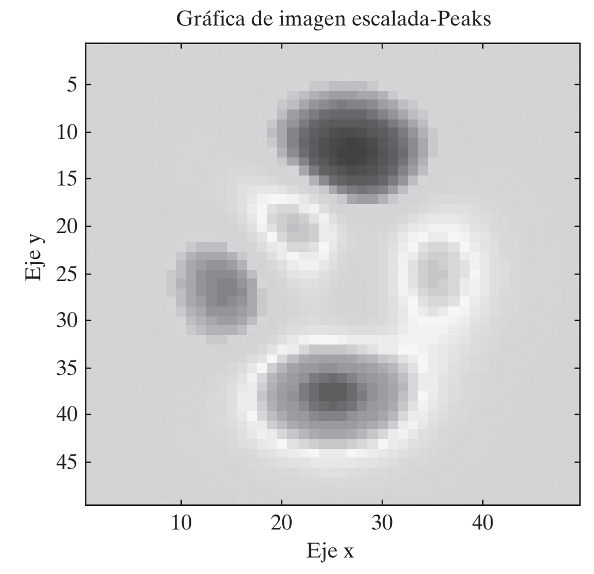


Figura 13.4
La función **peaks** representada con la función **imagesc**. Izquierda: las imágenes usualmente se representan partiendo de la esquina superior izquierda y se trabaja a través y hacia abajo, la forma en que se lee un libro. Derecha: la gráfica **pcolor** y la gráfica **imagesc** son imágenes espejo verticales una de otra.

Note que la imagen se voltea en comparación con la gráfica de seudocolor. Desde luego, en muchas aplicaciones gráficas no importa cómo se representan los datos, en tanto se comprenda la convención usada. Sin embargo, una fotografía que estuviera boca abajo en una imagen espejo vertical, obviamente no sería una representación aceptable.

13.1.1 Tipos de imagen

MATLAB reconoce tres diferentes técnicas para almacenar y representar imágenes:

Imágenes de intensidad (o escala de grises)
Imágenes indexadas
Imágenes RGB (o color verdadero)

Idea clave: existen dos funciones que se usan para desplegar imágenes: **imagesc** e **image**.

Imágenes de intensidad

Para crear la representación de la función **peaks** se usó una imagen de intensidad (figura 13.4) con la función imagen escalada (**imagesc**). En este enfoque, los colores en la imagen se determinan mediante un mapa de color. Los valores almacenados en la matriz de imagen están escalados, y los valores están correlacionados con un mapa conocido. (El mapa de color jet es por defecto.) Este enfoque funciona bien cuando el parámetro que se despliega no se correlaciona con un color real. Por ejemplo, la función **peaks** con frecuencia se compara con un rango de montaña y valle, ¿pero qué elevación es el color rojo? Es una elección arbitraria con base parcialmente en estética, pero también se pueden usar mapas de color para mejorar las características de interés en la imagen.

Considere este ejemplo: las imágenes en rayos X tradicionalmente se produjeron mediante la exposición de película fotográfica a radiación de rayos X. La mayoría de los rayos X de la actualidad se procesan como imágenes digitales y se almacenan en un archivo de datos, no hay película involucrada. Se puede manipular dicha película como se desee, pues la intensidad de la radiación de rayos X no corresponde a un color particular.

MATLAB incluye un archivo de muestra que es una fotografía digital de rayos X de una columna vertebral, adecuada para usarse con la función imagen escalada. Primero necesitará cargar el archivo:

```
load spine
```

El archivo cargado incluye algunas matrices (véase la ventana del área de trabajo), la matriz intensidad se llama **X**. Por tanto,

```
imagesc(X)
```

produce una imagen cuyos colores se determinan mediante el **colormap** actual, que por defecto es **jet**. Se regresa una representación que se parece más a unos rayos X tradicionales si se usa el colormap **bone**:

```
colormap(bone)
```

Esta imagen se muestra en la figura 13.5.

El archivo spine también incluye un colormap personalizado (custom), que resulta ser el colormap **bone**. Este arreglo se llama **map**. Los mapas de colores personalizados no son necesarios para desplegar imágenes de intensidad, y

```
colormap(map)
```

resulta en la misma imagen que se creó anteriormente.

Aunque es conveniente pensar en los datos de imagen como en una matriz, tales datos no necesariamente se almacenan de dicha forma en los formatos gráficos estándar. MATLAB incluye una función, imfinfo, que leerá archivos gráficos estándar y determinará qué tipo de

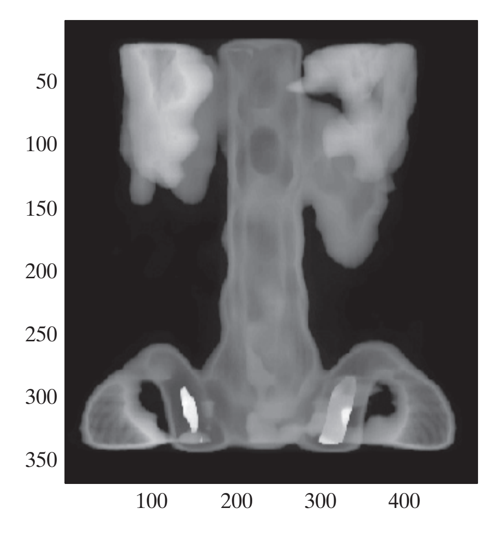


Figura 13.5
Rayos X digitales desplegados con el uso de la función **imagesc** y el colormap **bone**.

datos contiene el archivo. Considere el archivo mimas.jpg, que se descargó de Internet desde el website de la NASA (www.saturn.jpl.nasa.gov). El comando

```
imfinfo('mimas.jpg')
```

regresa la siguiente información (asegúrese de mencionar el nombre del archivo entre apóstrofes, esto es, como una cadena; además, note que la imagen es **'grayscale'**, otro término para una imagen de intensidad):

Idea clave: el esquema de color para una imagen se controla mediante el mapa de color.

```
ans =

        Filename: 'mimas.jpg'
        FileModDate: '06-Aug-2005 08:52:18'
        FileSize: 23459
        Format: 'jpg'
        FormatVersion: ''
        Width: 500
        Height: 525
        BitDepth: 8
        ColorType: 'grayscale'
        FormatSignature: ''
        NumberOfSamples: 1
        CodingMethod: 'Huffman'
        CodingProcess: 'Sequential'
        Comment: {'Created with The GIMP'}
```

Para crear una matriz MATLAB a partir de este archivo, se usa la función de lectura de imagen **imread** y se asignan los resultados a un nombre de variable, como **X**:

```
X=imread('mimas.jpg');
```

Entonces se puede graficar la imagen con la función **imagesc** y el colormap **gray**:

```
imagesc(X)
colormap(gray)
```

Los resultados se muestran en la figura 13.6a.

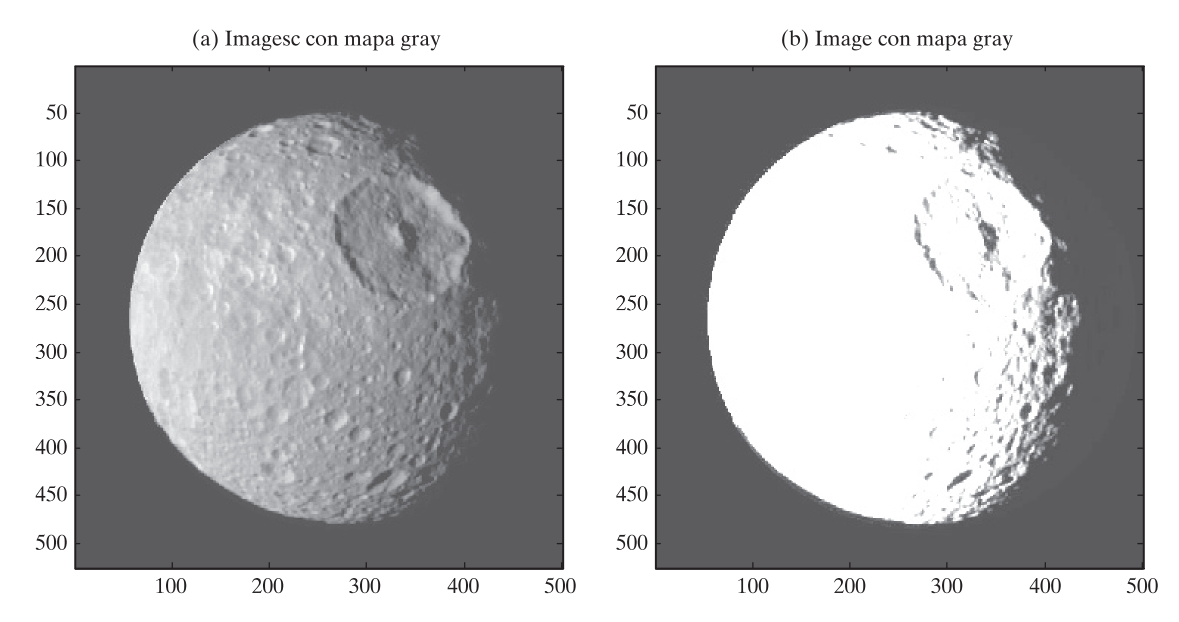


Figura 13.6
(a) Imagen de Mimas, una luna de Saturno, desplegada mediante la función imagen escalada, **imagesc**, y un mapa de color **gray**. (b) Imagen desplegada con la función imagen indexada, **image**, y un mapa de color **gray**.

Función imagen indexada
Cuando el color es importante, una técnica para crear una imagen se llama *imagen indexada*. En lugar de ser una lista de valores de intensidad, la matriz es una lista de colores. La imagen se crea en forma muy parecida a una pintura de "colorear por número". Cada elemento contiene un número que corresponde a un color. Los colores se mencionan en una matriz separada llamada colormap, que es una matriz $n \times 3$ que define n diferentes colores al identificar los componentes rojo, verde y azul de cada color. Para cada imagen se puede crear un colormap personalizado, o se podría usar un colormap interno.

Considere la imagen de muestra interna de un mandril, obtenido con

```
load mandrill
```

El archivo incluye una matriz indexada llamada **X** y un mapa de color llamado **map**. (Verifique la ventana del área de trabajo para confirmar que se cargaron dichos archivos; los nombres se usan comúnmente para imágenes guardadas desde un programa MATLAB.) La función **image** se usa para desplegar imágenes indexadas:

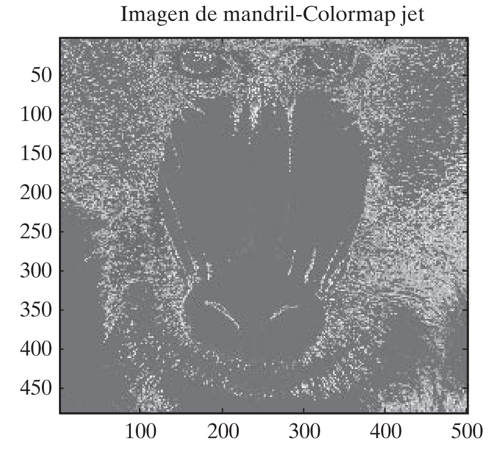
```
image(X)
colormap(map)
```

Las imágenes MATLAB se ajustan para llenar la ventana de figura, de modo que la imagen puede parecer distorsionada. Se puede forzar el despliegue del aspecto correcto con el uso del comando **axis**:

```
axis image
```

Los resultados se muestran en la figura 13.7.

Las funciones **image** e **imagesc** son similares, aunque pueden producir resultados muy diferentes. La imagen de Mimas en la figura 13.6b se produjo mediante la función **image** en lugar de la más apropiada función **imagesc**. El mapa de color **gray** no corresponde a los colores almacenados en la imagen de intensidad; el resultado es la imagen deslavada y carente de



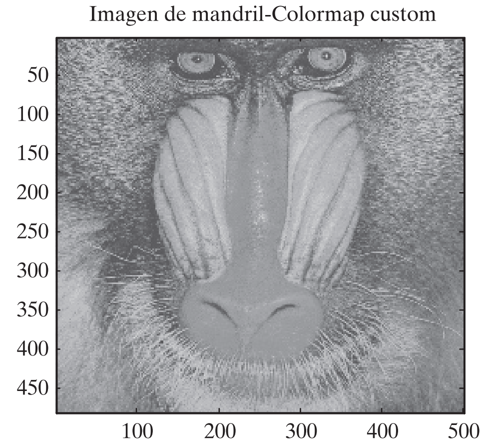


Figura 13.7
Izquierda: imagen de mandril antes de aplicar el mapa de color personalizado. Derecha: imagen de mandril con el mapa de color personalizado.

contraste. Es importante reconocer qué tipo de archivo despliega, de modo que pueda hacer la elección óptima de cómo representar la imagen.

Los archivos almacenados en el formato gráfico GIF con frecuencia se almacenan como imágenes indexadas. Esto puede no ser aparente cuando usa la función **imfinfo** para determinar los parámetros del archivo. Por ejemplo, la imagen en la figura 13.8 es parte de los clips art incluidos con Microsoft Word. La imagen se copió en el directorio actual y se usó **imfinfo** para determinar el tipo de archivo:

```
imfinfo('drawing.gif')

ans =

1x4 struct array with fields:
  Filename
  FileModDate
  FileSize
  Format   etc.
```



Figura 13.8
Clip art almacenado en el formato de archivo GIF.

Los resultados no dicen mucho, pero si hace doble clic en el nombre del archivo en el directorio actual, se lanza el Asistente de Importación —Import Wizard— (figura 13.9) y sugiere que se creen dos matrices: **cdata** y **colormap**. La matriz **cdata** es una matriz de imagen indexada, y **colormap** es el mapa de color correspondiente. En realidad, el nombre sugerido

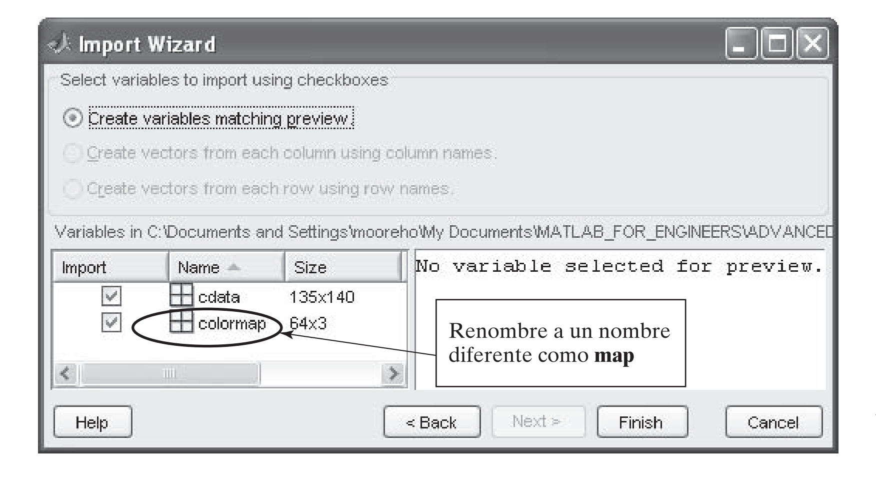


Figura 13.9
El Asistente de Importación se usa para crear una matriz de imagen indexada y un mapa de color a partir de un archivo GIF.

colormap es más bien extraño, porque si se le usa, reemplazará la función **colormap**. Necesitará renombrar esta matriz a algo diferente, como **map**, con el siguiente código:

```
image(cdata)
colormap(map)
axis image
axis off
```

Sugerencia

Existen algunas imágenes de muestra internas en MATLAB y almacenadas como imágenes indexadas. Puede acceder a estos archivos escribiendo

```
load <nombre de imagen>
```

Las imágenes disponibles son

```
flujet
durer
detail
mandrill
clown
spine
cape
earth
gatlin
```

Cada uno de estos archivos de imagen crea una matriz de valores índices llamada **X** y un mapa de color llamado **map**. Por ejemplo, para ver la imagen de la Tierra, escriba

```
load earth
image(X)
colormap(map)
```

También necesitará ajustar la tasa de aspecto del despliegue y remover el eje con los comandos

```
axis image
axis off
```

RGB: los colores primarios de la luz son rojo, verde y azul

Imágenes en color verdadero (RGB)

La tercera técnica para almacenar datos de imagen es en una matriz tridimensional, $m \times n \times 3$. Recuerde que una matriz tridimensional consta de filas, columnas y páginas. Los archivos de imagen de color verdadero constan de tres páginas, una por cada intensidad de color, rojo, verde o azul, como se muestra en la figura 13.10.

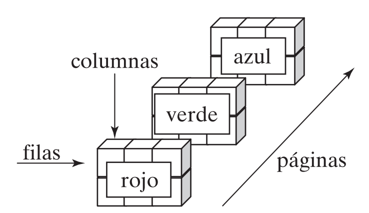


Figura 13.10
Las imágenes en color verdadero usan un arreglo multidimensional para representar en color de cada elemento.

Considere un archivo llamado **airplanes.jpg**. Puede copiar éste o un archivo similar (una imagen jpg coloreada) en su directorio actual para experimentar con imágenes de color verdadero. Puede usar la función **imfinfo** para determinar cómo el archivo airplanes almacena la imagen:

```
imfinfo('airplanes.jpg')

ans =

Filename: 'airplanes.jpg'
FileModDate: '12-Sep-2005 17:51:48'
FileSize: 206397
Format: 'jpg'
FormatVersion: "
Width: 1800
Height: 1200
BitDepth: 24
ColorType: 'truecolor'
FormatSignature: "
NumberOfSamples: 3
CodingMethod: 'Huffman'
CodingProcess: 'Sequential'
Comment: {}
```

Note que el tipo de color es **'truecolor'** y que el número de muestras es 3, lo que indica una página por cada intensidad de color.

Puede cargar la imagen con la función **imread** y desplegarla con la función **image**:

```
X=imread('airplanes.jpg');
image(X)
axis image
axis off
```

Note en la ventana del área de trabajo que **X** es una matriz $1200 \times 1800 \times 3$, una página por cada color. No necesita cargar un mapa de color, porque la información de intensidad de color se incluye en la matriz (figura 13.11).



Figura 13.11
Imagen de color verdadero de aviones. Toda la información de color se almacena en una matriz tridimensional. (Imagen usada con permiso del Dr. G. Jimmy Chen, Salt Lake Community College, Departamento de Ciencias de la Computación.)

EJEMPLO 13.1

Conjuntos Mandelbrot y Julia



Figura 13.12
Benoit Mandelbrot.

Benoit Mandelbrot (figura 13.12) es en gran medida responsable del actual interés en la geometría fractal. Su trabajo se construye en torno a los conceptos desarrollados por el matemático francés Gaston Julia en su artículo *Mémoire sur l'iteration des fonctions rationelles*, de 1919. Los avances en la obra de Julia tuvieron que esperar el desarrollo de las computadoras, y particularmente de las gráficas por computadora. En la década de 1970, Mandelbrot, entonces en IBM, revisó y profundizó en la obra de Julia y, de hecho, desarrolló algunos de los primeros programas de gráficos por computadora para desplegar los complicados y bellos patrones fractales que hoy llevan su nombre.

La imagen Mandelbrot se crea al considerar cada punto en el plano complejo, $x + yi$. Se hace $z(0) = x + yi$ y luego se itera de acuerdo con la siguiente estrategia:

$$z(0) = x + yi$$
$$z(1) = z(0)^2 + z(0)$$
$$z(2) = z(1)^2 + z(0)$$
$$z(3) = z(2)^2 + z(0)$$
$$z(n) = z(n - 1)^2 + z(0)$$

La serie parece converger o evitar el infinito. El conjunto Mandelbrot está compuesto de los puntos que convergen. Las hermosas imágenes que probablemente usted ha visto fueron creadas al contar cuántas iteraciones fueron necesarias para que el valor de z en un punto superara cierto valor umbral, con frecuencia la raíz cuadrada de 5. Se supone, aunque no se ha podido probar, que si dicho umbral se alcanza, la serie continuará para divergir y eventualmente tender al infinito.

1. Establezca el problema.
   Escribir un programa MATLAB para desplegar el conjunto Mandelbrot.
2. Describa las entradas y salidas.

   Entrada Se sabe que el conjunto Mandelbrot se encuentra en alguna parte del plano complejo y que

$$-1.5 \leq x \leq 1.0$$
$$-1.5 \leq y \leq 1.5$$

   También se sabe que se puede describir cada punto en el plano complejo como

$$z = x + yi$$

3. Desarrolle un ejemplo a mano.
   Trabaje las primeras iteraciones para un punto que se espera converja, como ($x = -0.5$, $y = 0$):

$$z(0) = 0.5 + 0i$$
$$z(1) = z(0)^2 + z(0) = (-0.5)^2 - 0.5 = 0.25 - 0.5 = -0.25$$
$$z(2) = z(1)^2 + z(0) = (-0.25)^2 - 0.5 = 0.0625 - 0.5 = -.4375$$
$$z(3) = z(2)^2 + z(0) = (-0.4375)^2 - 0.5 = 0.1914 - 0.5 = -.3086$$
$$z(4) = z(3)^2 + z(0) = (-.3086)^2 + 0.5 = 0.0952 - 0.5 = -.4048$$

Parece como que esta secuencia converge a un punto en torno a −0.4. (Como ejercicio, podría crear un programa MATLAB para calcular los primeros 20 términos de la serie y graficarlos.)

4. Desarrolle una solución MATLAB.

```
%Ejemplo 13.1   Imagen Mandelbrot
clear, clc
iterations=80;
grid_size = 500;
[x,y]=meshgrid(linspace(-1.5,1.0,grid_size),linspace
  (-1.5,1.5,grid_size));
c = x+i*y;
z=zeros(size(x));         % establezca la matriz inicial a 0
map=zeros(size(x));       % cree un mapa de todos los puntos
                          % retícula igual a 0
for k=1:iterations
z=z.^2 +c;
a=find(abs(z)>sqrt(5));   %Determine cuáles elementos
                          exceden sqrt(5)
map(a)=k;
end
figure(1)
image(map)           %Cree una imagen
colormap(jet)
```

La imagen producida se muestra en la figura 13.13.

5. Ponga a prueba la solución.
Se sabe que todos los elementos en la región coloreada sólida de la imagen (azul oscuro si observa la imagen en una pantalla de computadora) estarán por abajo de la raíz cuadrada de 5. Una forma alternativa de examinar los resultados es crear una imagen con base en dichos valores en lugar del número de iteraciones necesarias para superar el umbral. Necesitará multiplicar cada valor por un múltiplo común con la finalidad de lograr alguna variación de color. (De otro modo los valores estarán muy cercanos unos de otros.) El código MATLAB es el que sigue:

```
figure(2)
multiplier=100;
map=abs(z)*multiplier;
image(map)
```

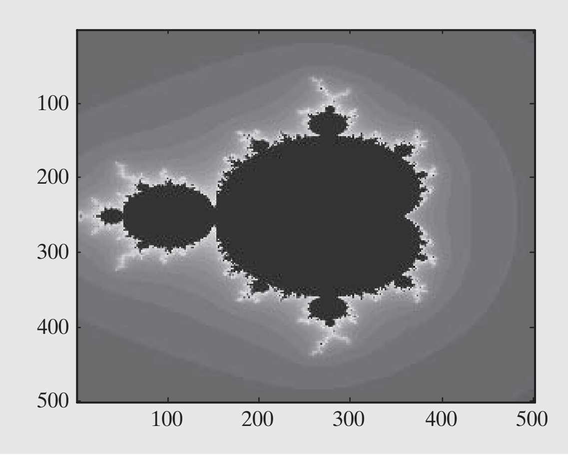


Figura 13.13
Imagen Mandelbrot. La figura se creó al determinar cuántas iteraciones se requirieron para que los valores de elemento calculados superaran la raíz cuadrada de 5.

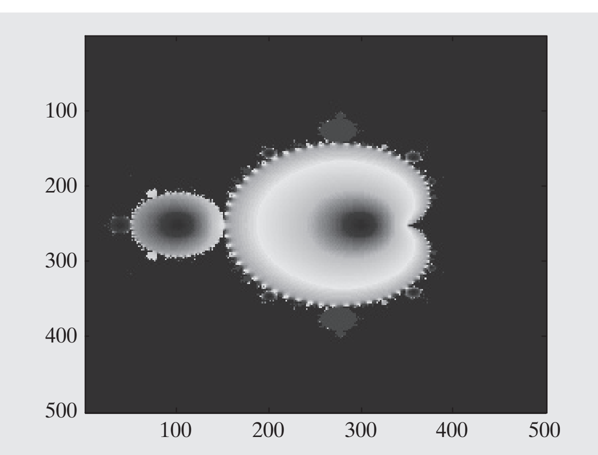


Figura 13.14
Imagen basada en el conjunto Mandelbrot, que muestra cómo varían los miembros del conjunto. La estructura realmente interesante está en la frontera del conjunto.

Los resultados se muestran en la figura 13.14.

Ahora que se creó una imagen de todo el conjunto Mandelbrot, sería interesante observar más de cerca algunas de las estructuras en la frontera. Al agregar las siguientes líneas de código al programa se puede acercar repetidamente a cualquier punto en la imagen:

```
cont=1;
while(cont==1)
figure(1)
disp('Ahora acérquese')
disp('Mueva el cursor a la esquina superior izquierda
  del área que quiere expandir')
[y1,x1]=ginput(1);
disp('Mueva a la esquina inferior derecha del área que
  quiere expandir')
[y2,x2]=ginput(1);
xx1=x(round(x1),round(y1));
yy1=y(round(x1),round(y1));
xx2=x(round(x2),round(y2));
yy2=y(round(x2),round(y2));
%%
[x,y]=meshgrid(linspace(xx1,xx2,grid_size),linspace(yy1,
  yy2,grid_size));
c = x+i*y;
z=zeros(size(x));
map=zeros(size(x));
for k=1:iterations
  z=z.^2 +c;
  a=find(abs(z)>sqrt(5));
  map(a)=k;
end
image(map)
colormap(jet)
```

```
again = menu('¿Quiere acercarse de nuevo? ','Sí','No');
switch again
  case 1
    cont=1;
  case 2
    cont=0;

end
end
```

La figura 13.15 muestra algunas de las imágenes creadas al recalcular con áreas cada vez más pequeñas.

Puede experimentar tanto con la función **image** como con **imagesc** y observar cómo difieren las imágenes. Intente también algunos mapas de color diferentes.

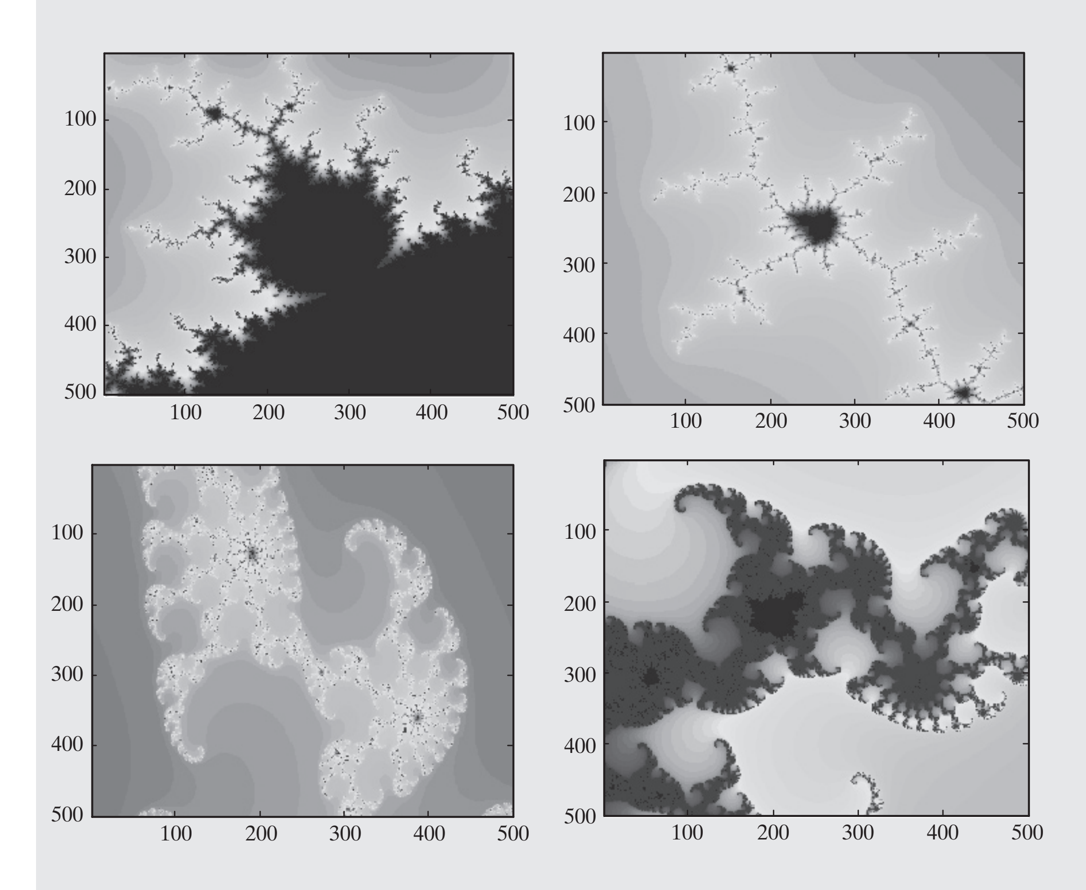


Figura 13.15
Imágenes creadas al acercarse sobre el conjunto Mandelbrot desde un programa MATLAB.

13.1.2 Lectura y escritura de archivos de imagen

Se introdujeron funciones para leer archivos de imagen conforme se exploraron las tres técnicas de almacenamiento de información de imagen. MATLAB también incluye funciones para escribir imágenes creadas por el usuario en cualquiera de varios formatos. En esta sección se explorarán con más detalle estas funciones de lectura y escritura.

Lectura de información de imagen

Probablemente la forma más sencilla de leer información de imagen en MATLAB es sacar ventaja del Asistente de Importación interactivo. En la ventana del directorio actual, simplemente haga doble clic en el nombre de archivo de la imagen a importar. MATLAB sugerirá nombres de variable adecuados y pondrá las matrices a disposición para una observación previa en la ventana de edición (figura 13.9).

El problema con la importación interactiva de cualquier dato es que usted no puede incluir las instrucciones en un programa MATLAB; para ello, necesita usar una de las funciones de importación. Para la mayoría de los formatos estándar de imagen, como jpg o tif, la técnica apropiada es la función **imread** descrita en la sección anterior. Si el archivo es un archivo **.mat** o uno **.dat**, la forma más sencilla de importar los datos es usar la función **load**:

```
load <nombre_de_archivo>
```

Para archivos **.mat**, ni siquiera necesita incluir la extensión **.mat**. Sin embargo, necesitará incluir la extensión para un archivo **.dat**:

```
load <nombre_de_archivo.dat>
```

Ésta es la técnica que se usó para cargar los archivos de imagen internos descritos anteriormente. Por ejemplo,

```
load cape
```

importa la matriz de imagen y el mapa de color en el directorio actual, y entonces se pueden usar los comandos

```
image(X)
colormap(map)
load cape
image(X)
colormap(map)
axis image
axis off
```

para crear la imagen, que se muestra en la figura 13.16.



Figura 13.16
Imagen creada al cargar un archivo interno.

Almacenamiento de información de imagen

Puede guardar una imagen que haya creado en MATLAB de la misma forma en que guarda cualquier figura. Seleccione

File → Save As...

y elija el tipo de archivo y la ubicación donde le gustaría guardar la imagen. Por ejemplo, para guardar la imagen del conjunto Mandelbrot creada en el ejemplo 13.1 y que se muestra en la figura 13.13, tal vez quiera especificar un metarchivo mejorado (**.emf**), como se muestra en la figura 13.17.

También podría guardar el archivo al usar la función **imwrite**. Esta función acepta algunas entradas diferentes, dependiendo del tipo de datos que le gustaría almacenar.

Por ejemplo, si tiene un arreglo de intensidad (escala de grises) o un arreglo color verdadero (RGB), la función **imwrite** espera entrada de la forma

```
imwrite(arrayname,'filename.format')
```

donde

arrayname es el nombre del arreglo MATLAB en el que se almacenan los datos,
filename es el nombre que quiere usar para almacenar los datos, y
format es la extensión del archivo, como jpg o tif.

Por tanto, para almacenar una imagen RGB en un archivo jpg llamado flowers, el comando sería

```
imwrite(X,'flowers.jpg')
```

(Consulte los archivos **help** para una lista de formatos gráficos soportados por MATLAB.)

Si tiene una imagen indexada (una imagen con un mapa de color personalizado), necesitará almacenar tanto el arreglo como el mapa de color:

```
imwrite(arrayname, colormap_name,'filename.format')
```

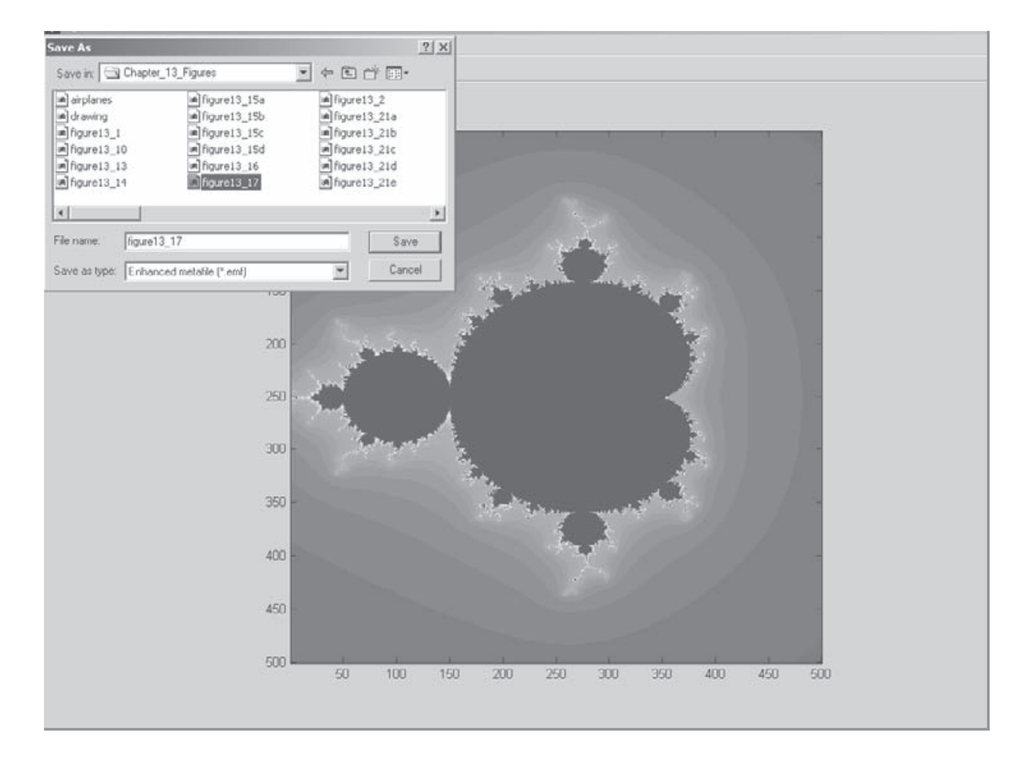


Figura 13.17
Esta imagen de un conjunto Mandelbrot se guardará como un metarchivo mejorado.

En el caso del conjunto Mandelbrot, se necesitaría guardar el arreglo y el mapa de color usado para seleccionar los colores en la imagen:

```
imwrite(map,jet,'my_mandelbrot.jpg')
```

13.2 MANIPULACIÓN DE GRÁFICOS

handle: apodo

Un handle es un apodo dado a un objeto en MATLAB. Una descripción completa de los sistemas gráficos usados en MATLAB resulta complicada y está más allá del ámbito de este texto. (Para más detalles, remítase al tutorial **help** de MATLAB.) Sin embargo, se dará una breve introducción a los gráficos handle y luego se ilustrarán algunos de sus usos.

MATLAB usa un sistema jerárquico para crear gráficos (figura 13.18). El objeto básico de graficación es la figura. La figura puede contener varios objetos diferentes, incluido un conjunto de ejes. Piense en los ejes como puestos en capas en lo alto de la ventana de figura. Los ejes también pueden contener algunos objetos diferentes, incluida una gráfica como la que se muestra en al figura 13.19. De nuevo, considere a la gráfica como en capas arriba de los ejes.

Cuando usa una función **plot**, desde la ventana de comandos o desde un programa en archivo-m, MATLAB automáticamente crea una figura y un eje adecuado y luego dibuja la gráfica sobre el eje. MATLAB usa valores por defecto para muchas de las propiedades del objeto graficado. Por ejemplo, la primera línea dibujada siempre es azul a menos que el usuario lo cambie específicamente.

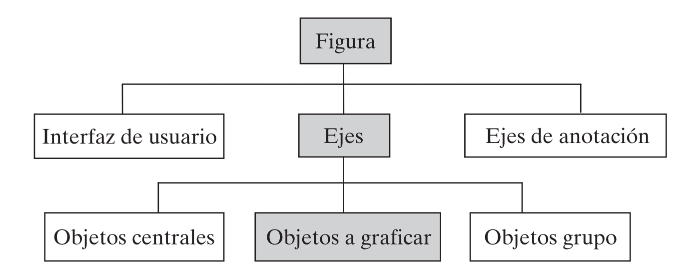


Figura 13.18
MATLAB usa un sistema jerárquico para organizar información de graficación, como se muestra en esta representación del menú help de MATLAB.

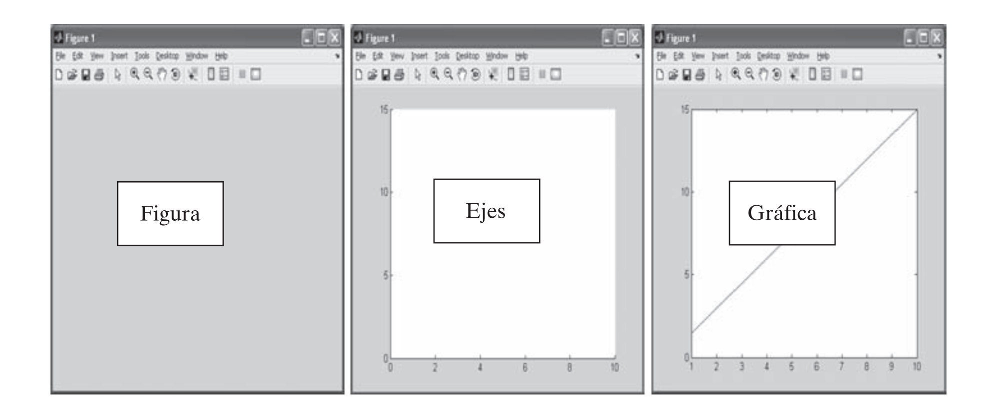


Figura 13.19
Anatomía de una gráfica. Izquierda: las ventanas de figuras se usan para muchas cosas, incluidas interfaces gráficas de usuario y gráficas. Para crear una gráfica necesita una ventana de figura. Centro: antes de poder dibujar una gráfica en esta ventana de figura, necesitará un conjunto de ejes para dibujar sobre ellos. Derecha: una vez que sepa dónde están los ejes y cuáles son las propiedades de los ejes (como el espaciamiento), puede dibujar el gráfico.

13.2.1 Handles de gráficas

Asignar un nombre a una gráfica (llamado handle) le permite pedir fácilmente a MATLAB una lista de las propiedades del objeto graficado. Por ejemplo, cree la gráfica simple que se muestra en la figura 13.19 y asígnele un handle:

```
x=1:10;
y=x.*1.5;
h=plot(x,y)
```

La variable **h** es el handle de la gráfica. (Se podría haber escogido cualquier nombre de variable.) Ahora se puede usar la función **get** para preguntar a MATLAB las propiedades de la gráfica:

```
get(h)
```

La función regresa una lista completa de las propiedades que representan la línea que se dibujó en el eje, las cuales fueron colocadas en la ventana de figura:

```
Color: [0 0 1]
EraseMode: 'normal'
LineStyle: '-'
LineWidth: 0.5000
Marker: 'none'
MarkerSize: 6
MarkerEdgeColor: 'auto'
MarkerFaceColor: 'none'
XData: [1 2 3 4 5 6 7 8 9 10]
YData: [1.5000 3 4.5000 6 7.5000 9 10.5000 12 13.5000 15]
ZData: [1x0 double]
                    .
                    .
                    .
```

Note que la propiedad color se menciona como [0 0 1]. Los colores se describen como intensidades de cada uno de los colores primarios de luz: rojo, verde y azul. El arreglo [0 0 1] dice que no hay rojo, no hay verde y es 100% azul.

13.2.2 Handles de figura

También se puede especificar un nombre handle para la ***ventana de figuras***. Dado que esta gráfica se dibujó en la ventana de figura llamada figure 1, el comando sería

```
f_handle=figure(1)
```

Usar el comando **get** regresa resultados similares:

```
get(f_handle)
Alphamap = [ (1 by 64) double array]
BackingStore = on
CloseRequestFcn = closereq
Color = [0.8 0.8 0.8]
Colormap = [ (64 by 3) double array]
CurrentAxes = [150.026]
CurrentCharacter =
CurrentObject = []
CurrentPoint = [240 245]
DockControls = on
```

```
DoubleBuffer = on
FileName = [ (1 by 96) char array]
                                          .
                                          .
                                          .
```

Note que las propiedades son diferentes de las anteriores. En particular el color (que es el color de fondo de la ventana) es [0.8, 0.8, 0.8], que especifica intensidades iguales de rojo, verde y azul, lo que, desde luego, resulta en un fondo blanco.

Si no se especificó un nombre handle, se puede pedir a MATLAB que determine la figura actual con el comando **gcf** (get current figure: obtener figura actual),

```
get(gcf)
```

que da los mismos resultados.

13.2.3 Handles de ejes

Así como se puede asignar un handle a la ventana de figura y la gráfica misma, se puede asignar un handle al eje mediante la función **gca** (get current axis: obtener eje actual):

```
h_axis = gca;
```

Usar este handle con el comando **get** le permite ver las propiedades del eje:

```
get(h_axis)
ActivePositionProperty = outerposition
ALim = [0.1 10]
ALimMode = auto
AmbientLightColor = [1 1 1]
Box = off
CameraPosition = [-1625.28 -2179.06 34.641]
CameraPositionMode = auto
CameraTarget = [201 201 0]
                                          .
                                          .
                                          .
```

13.2.4 Anotación de ejes

Además de los tres componentes descritos en las secciones anteriores, otra capa transparente se agrega a la gráfica. Esta capa se usa para insertar objetos de anotación, como líneas, leyendas y recuadros de texto, en la figura.

13.2.5 Uso de handles para manipular gráficos

Entonces, ¿qué se puede hacer con toda esta información? Se puede usar la función **set** para cambiar las propiedades del objeto. La función **set** requiere el handle de objeto en el primer campo de entrada y luego cadenas alternadas que especifiquen un nombre de propiedad, seguidos por un nuevo valor. Por ejemplo,

```
set(h,'color','red')
```

le dice a MATLAB que vaya a la gráfica llamada **h** (no la figura, sino el dibujo real) y cambie el color a rojo. Si quiere cambiar alguna de las propiedades de la figura, puede hacerlo de esta forma con el nombre handle de figura o con la función **gcf**. Por ejemplo, para cambiar el nombre de la figura 1, use el comando

```
set(f_handle,'name', 'My Graph')
```

o

```
set(gcf,'name', 'My Graph')
```

Puede lograr interactivamente lo mismo al seleccionar **View** de la barra de menú figura y elegir el editor de propiedad:

View → Property Editor

Puede acceder a todas las propiedades si elige Property Inspector desde la ventana desplegable del editor de propiedad (figura 13.20). Explorar la ventana del inspector de propiedad es una buena forma de encontrar cuáles propiedades están disponibles para cada objeto de gráficos.

13.3 ANIMACIÓN

Existen dos técnicas para crear una animación en MATLAB:

volver a dibujar y borrar

crear una película

En cada caso se usan gráficos handle para crear la animación.

13.3.1 Volver a dibujar y borrar

Para crear una animación mediante volver a dibujar y borrar, primero cree una gráfica y luego ajuste las propiedades de la gráfica cada vez mediante un bucle. Considere el siguiente ejemplo: puede definir un conjunto de parábolas con la ecuación

$$y = kx^2 - 2$$

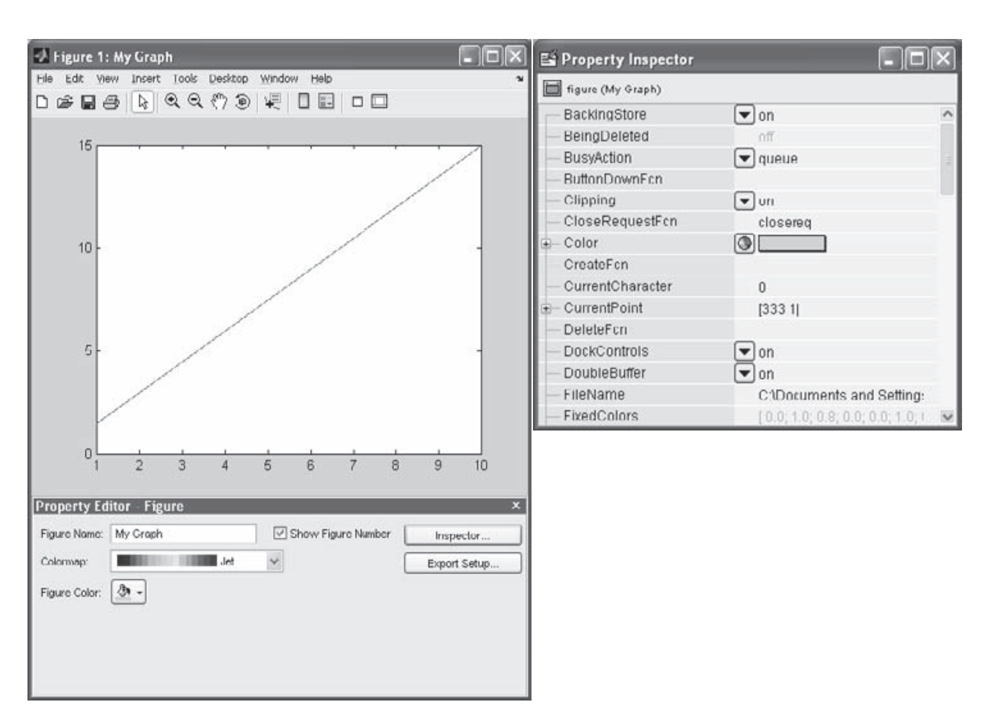


Figura 13.20
Edición interactiva de propiedades.

Cada valor de *k* define una parábola diferente. Podría representar los datos con una gráfica tridimensional; sin embargo, otro enfoque sería crear una animación en la que se dibuje una serie de gráficas, cada una con un valor diferente de *k*. el código para crear dicha animación es el siguiente:

```
clear,clc,clf
x=-10:0.01:10;       % Defina los valores x
k=-1;                % Establezca un valor inicial de k
y=k*x.^2-2;          % Calcule el primer conjunto de valores y
h=plot(x,y);         % Cree la figura y asigne
                     % un handle a la gráfica
grid on
%set(h,'EraseMode','xor')    % La animación corre más rápido
                             % si activa esta línea
axis([-10,10,-100,100])      % Especifique los ejes
while k<1                    % Comience un bucle
  k=k + 0.01;                % Incremente k
  y=k*x.^2-2;                % Recalcule y
  set(h,'XData',x,'YData',y)    % Reasigne los valores
                                % x y y usados en la gráfica
  drawnow    % Vuelva a dibujar la gráfica ahora - no espere
             % hasta que el programa termine de correr
end
```

En este ejemplo se usaron gráficos handle para volver a dibujar la gráfica cada vez mediante el bucle, en lugar de crear una nueva ventana de figura cada vez que pasa por el bucle. Además, se usaron los objetos **XData** y **YData** de la gráfica. Estos objetos asignan los puntos de datos a graficar. Usar la función **set** le permite especificar nuevos valores de **x** y **y** y crear una gráfica diferente cada vez que se llama la función **drawnow**. En la figura 13.21 se muestra una selección de los marcos creados por el programa y usados en la animación.

En el programa, note la línea

```
%set(h,'EraseMode','xor')
```

Si activa esta línea al remover el operador comentario (%), el programa no borra toda la gráfica cada vez que se vuelve a dibujar la gráfica. Sólo cambian los pixeles que cambian color. Esto hace que la animación corra más rápido, una característica que es importante cuando la gráfica es más complicada que la simple parábola usada en este ejemplo.

Remítase al tutorial **help** para una animación de muestra que modela movimiento browniano.

13.3.2 Películas

Animar la película de una línea no es computacionalmente intenso y es fácil obtener un movimiento suave y agradable. Considere este código que produce animación de gráfica de superficie más complicada:

```
clear,clc
x=0:pi/100:4*pi;
y= x;
[X,Y]=meshgrid(x,y);
```

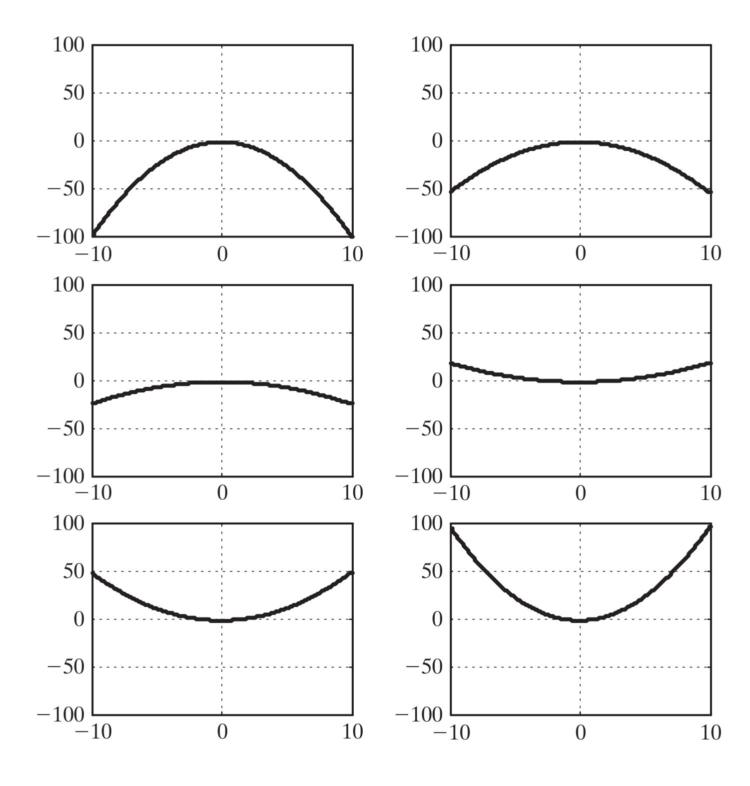


Figura 13.21
La animación funciona al volver a dibujar la gráfica varias veces.

```
z=3*sin(X)+ cos(Y);
h=surf(z);
axis tight
set(gca,'nextplot','replacechildren');
%Diga al programa que sustituya la superficie cada vez,
mas no el eje
shading interp
colormap(jet)
for k=0:pi/100:2*pi
  z=(sin(X) + cos(Y)).*sin(k);
  set(h,'Zdata',z)
  drawnow
end
```

En la figura 13.22 se presenta un cuadro de muestra de esta animación.

Si tiene una computadora rápida, la animación parecerá suave. Sin embargo, en una computadora más lenta puede ver movimiento espasmódico y pausas mientras el programa crea cada nueva gráfica. Para evitar este problema, puede crear un programa que capture cada "cuadro" y luego, una vez que todos los cálculos están hechos, reproducir los cuadros como una película.

```
clear,clc
x=0:pi/100:4*pi;
y= x;
[X,Y]=meshgrid(x,y);
z=3*sin(X)+ cos(Y);
h=surf(z);
```

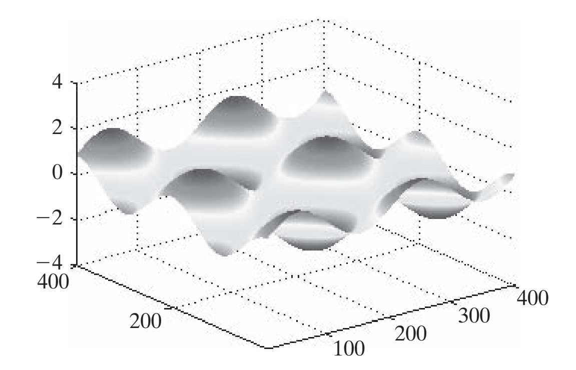


Figura 13.22
La animación de esta figura se mueve arriba y abajo en movimiento similar.

```
axis tight
set(gca,'nextplot','replacechildren');
shading interp
colormap(jet)
m=1;
for k=0:pi/100:2*pi
  z=(sin(X) + cos(Y)).*sin(k);
  set(h,'Zdata',z)
  M(m)=getframe;        %Crea y guarda cada cuadro
                        %de la película
  m=m+1;
end
  movie(M,2)            %Reproduce la película dos veces
```

Idea clave: las películas registran una animación para reproducción posterior.

Cuando corre este programa, en realidad verá la película cuatro veces: una vez cuando se crea, una vez cuando se carga en el “reproductor de películas” y las dos veces especificadas en la función **movie**. Una ventaja de este enfoque es que puede reproducir la película de nuevo sin volver a hacer los cálculos, pues la información se almacena (en el ejemplo) en el arreglo llamado **M**. Note en la ventana del área de trabajo —Workspace— (figura 13.23) que **M** es un arreglo estructura moderadamente grande (~90 MB).

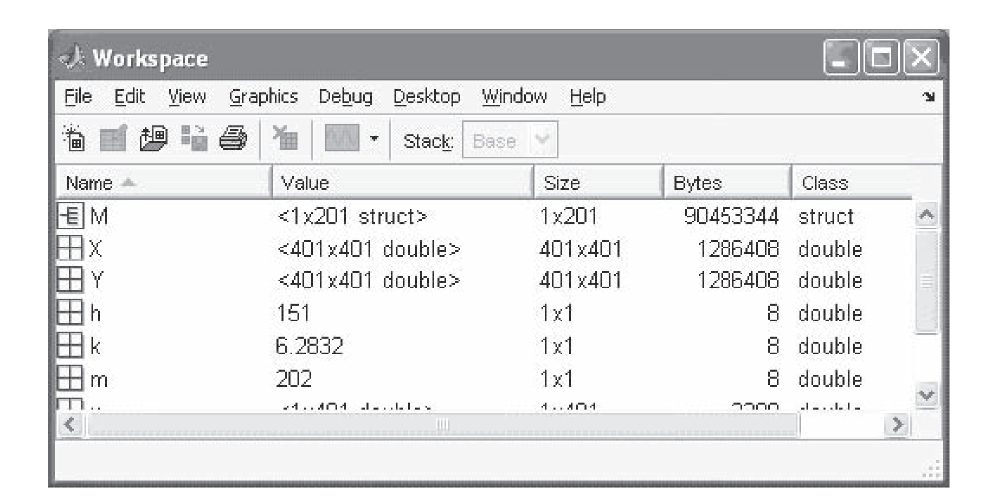


Figura 13.23
Las películas se guardan en un arreglo estructura, como el arreglo **M** que se muestra en esta figura.

EJEMPLO 13.2

Una película Mandelbrot

Los cálculos que se requieren para crear una imagen Mandelbrot necesitan significativos recursos computaciones y pueden tardar varios minutos. Si quiere acercarse en un punto de la imagen Mandelbrot, una elección lógica es hacer los cálculos y crear una película, que se puede ver más tarde. En este ejemplo, comience con el programa en archivo-m MATLAB descrito por primera vez en el ejemplo 13.1 y cree una película de 100 cuadros.

1. Establezca el problema.
   Cree una película mediante acercamiento en un conjunto Mandelbrot.
2. Describa las entradas y salidas.

   Entrada La imagen Mandelbrot completa descrita en el ejemplo 13.1

   Salida Una película de 100 cuadros

3. Desarrolle un ejemplo a mano.
   Un ejemplo a mano no tiene sentido para este problema, pero lo que se puede hacer es crear un programa con un pequeño número de iteraciones y elementos para probar la solución y luego usarlo para crear una secuencia más detallada que sea computacionalmente más intensa. He aquí el primer programa:

```
%Ejemplo 13.2   Imagen Mandelbrot
%  La primera parte de este programa es la misma que la del
ejemplo 13.1
clear, clc
iterations=20;          % Limite el número de iteraciones en
                        % este primer paso
grid_size = 50;         % Use una pequeña retícula para hacer
                        % que el programa corra más rápido
X=linspace(-1.5,1.0,grid_size);
Y=linspace(-1.5,1.5,grid_size);
[x,y]=meshgrid(X,Y);
c = x+i*y;
z=zeros(size(x));
map=zeros(size(x));
for k=1:iterations
  z=z.^2 +c;
  a=find(abs(z)>sqrt(5));
  map(a)=k;
end
figure(1)
h=imagesc(map)

%% Nueva sección de código

N(1)=getframe;     %Obtenga el primer cuadro de la película
disp('Ahora acérquese')
disp('Mueva el cursor a un punto donde le gustaría acercarse')
[y1,x1]=ginput(1)       %Seleccione el punto para acercarse

xx1=x(round(x1),round(y1))
yy1=y(round(x1),round(y1))

%%
for k=2:100  %Calcule y despliegue las nuevas imágenes
  k    %Envíe el número de iteración a la ventana de comandos
```

```
[x,y]=meshgrid(linspace(xx1-1/1.1^k,xx1+1/1.1^k,grid_size),...
     linspace(yy1-1/1.1^k,yy1+1/1.1^k,grid_size));
c = x+i*y;
z=zeros(size(x));
map=zeros(size(x));
for j=1:iterations
  z=z.^2 +c;
  a=find(abs(z)>sqrt(5));
  map(a)=j;
end
set(h,'CData',map)    % Recupere los datos de imagen desde
                      % el mapa de variables
colormap(jet)
N(k)=getframe;        % Capture el cuadro actual
end
movie(N,2)            % Reproduzca la película dos veces
```

Esta versión del programa corre más rápidamente y regresa imágenes de baja resolución (figura 13.24) lo que demuestra que el programa funciona.

4. Desarrolle una solución MATLAB.
La versión final del programa se crea al cambiar sólo dos líneas de código:

```
iterations=80;        % Aumente el número de iteraciones
grid_size = 500;      % Use una gran retícula para ver más
                        detalle
```

Esta versión "completa" del programa toma aproximadamente media hora para correr en un procesador Pentium de 2.0 GHz con 1.0 GB de RAM. Los cuadros seleccionados se muestran en la figura 13.25. Desde luego, el tiempo que tarde en su computadora será mayor o menor, dependiendo de los recursos de su sistema. Un ciclo de la película creada por el programa se reproduce en aproximadamente 10 segundos.

5. Ponga a prueba la solución.
Pruebe el programa varias veces y observe las imágenes creadas cuando se acerca a diferentes porciones del conjunto Mandelbrot. Puede experimentar con creciente número de iteraciones para crear la imagen y con el mapa de colores.

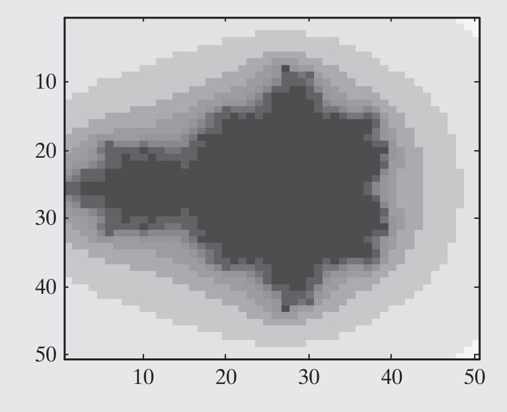


Figura 13.24
Imagen Mandelbrot de baja resolución.

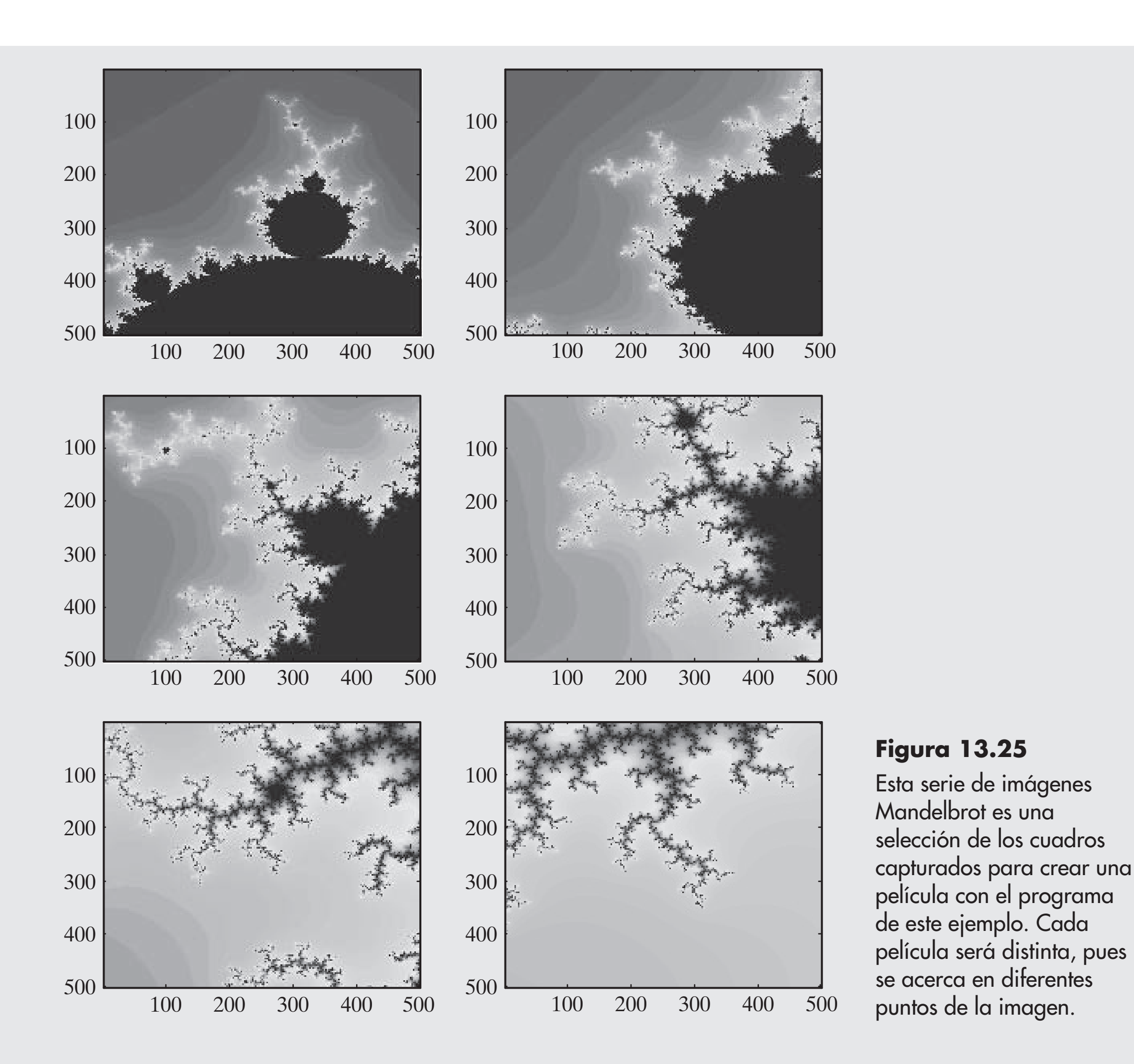


Figura 13.25
Esta serie de imágenes Mandelbrot es una selección de los cuadros capturados para crear una película con el programa de este ejemplo. Cada película será distinta, pues se acerca en diferentes puntos de la imagen.

13.4 OTRAS TÉCNICAS DE VISUALIZACIÓN

13.4.1 Transparencia

Cuando se representan superficies en MATLAB, se usa un esquema de coloración opaco. Este enfoque es estupendo para muchas superficies, pero puede oscurecer detalles de otras. Tome, por ejemplo, esta serie de comandos que crean dos esferas, una dentro de la otra:

```
clear,clc,clf        % Limpie la ventana de comandos y la
                     % ventana de figura actual
n = 20;              % Defina la superficie de una esfera,
                     % mediante coordenadas polares
Theta = linspace(-pi,pi,n);
Phi = linspace(-pi/2,pi/2,n);
[theta,phi]=meshgrid(Theta,Phi);
X = cos(phi).*cos(theta);    % Traduzca al sistema
                             % coordenado xyz
Y = cos(phi).*sin(theta);
Z = sin(phi);
```

```
surf(X,Y,Z)  %Cree una gráfica de superficie de una esfera
             %de radio 1
axis square
axis([-2,2,-2,2,-2,2]) %Especifique el tamaño de eje
hold on
pause                  %Pause el programa
surf(2*X,2*Y,2*Z)      %Agregue una segunda esfera de radio 2
pause                  %Pause el programa
alpha(0.5)             %Establezca nivel de transparencia
```

La esfera interior está oculta por la esfera exterior hasta que se emite el comando de transparencia,

```
alpha(0.5)
```

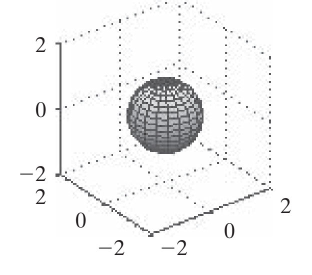
que establece el nivel de transparencia. Un valor de 1 corresponde a opaco y 0 a completamente transparente. Los resultados se muestran en la figura 13.26.

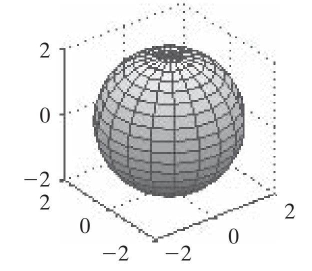
La transparencia se puede agregar a superficies, imágenes y objetos parche.

13.4.2 Líneas ocultas

Cuando se crean gráficas de malla, cualquier parte de la superficie que esté oscurecida no se dibuja. Por lo general, esto hace que la gráfica sea más sencilla de interpretar. Las dos esferas que se muestran en la figura 13.27 se crearon con el uso de las coordenadas **X**, **Y** y **Z** calculadas en la sección anterior. Aquí están los comandos MATLAB:

```
figure(3)
subplot(1,2,1)
mesh(X,Y,Z)
axis square
subplot(1,2,2)
```





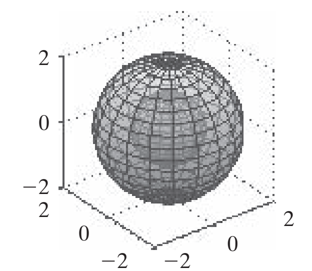
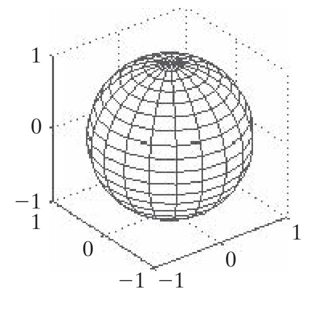


Figura 13.26
Agregar transparencia a una gráfica de superficie posibilita ver detalles ocultos.



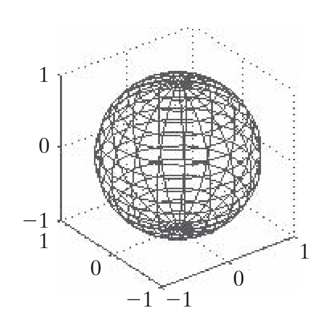


Figura 13.27
Izquierda: las gráficas de malla no muestran líneas de malla que se oscurecerían por una figura sólida. Derecha: el comando **hidden off** fuerza al programa a dibujar las líneas ocultas.

```
mesh(X,Y,Z)
axis square
hidden off
```

El valor por defecto para el comando **hidden** es **on**, que produce gráficas de malla en las que las líneas oscuras se ocultan automáticamente, como se muestra a la izquierda de la figura 13.27. Emitir el comando **hidden off** da los resultados que se muestran a la derecha de la figura 13.27.

13.4.3 Iluminación

Idea clave: la función **camlight** le permite ajustar la iluminación de la figura.

MATLAB incluye extensas técnicas para manipular la iluminación que se usa para representar gráficas de superficie. La posición de la luz virtual se puede cambiar e incluso manipular durante animaciones. La barra de herramientas figura incluye iconos que le permiten ajustar interactivamente la iluminación, de modo que puede obtener justo el efecto que quiera. Sin embargo, la mayoría de las gráficas realmente necesitan que la iluminación sólo se apague o encienda, lo que se logra con la función **camlight**. (Por defecto es apagado.) La figura 13.28 muestra los resultados que se logran cuando **camlight** se enciende sobre una esfera simple. El código a usar es

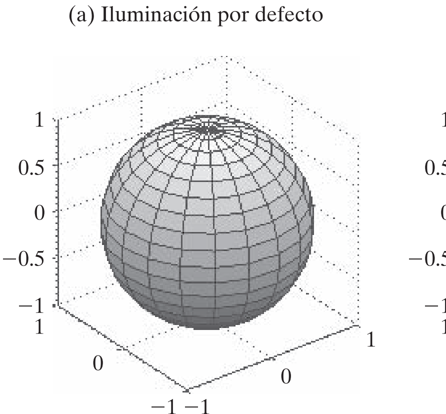
```
Sphere
camlight
```

La posición por defecto para la luz es arriba y a la derecha de la "cámara". Las opciones incluyen las siguientes:

camlight right	arriba y a la derecha de la cámara (por defecto)
camlight left	arriba y a la izquierda de la cámara
camlight headlight	colocada sobre la cámara
camlight(azimuth,elevation)	usted determina la posición de la luz
camlight('infinite')	modela una fuente de luz ubicada en el infinito (como el Sol)

13.5 INTRODUCCIÓN A VISUALIZACIÓN DE VOLUMEN

MATLAB incluye algunas técnicas de visualización que le permiten analizar datos recopilados en tres dimensiones, como la rapidez del viento medida en varias ubicaciones y elevacio-



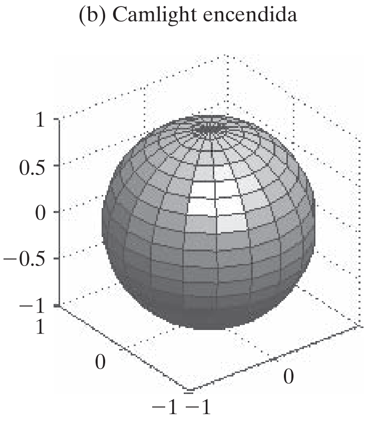


Figura 13.28
(a) La iluminación por defecto es difusa.
(b) Cuando se emite el comando **camlight**, se modela un reflector, ubicado en la posición de la cámara.

nes. También le permite visualizar los resultados de los cálculos realizados con tres variables, como $y = f(x, y, z)$. Estas técnicas de visualización caen en dos categorías:

- visualización de volumen de datos escalares (donde los datos recopilados o calculados son un solo valor en cada punto, como la temperatura).
- visualización de volumen de datos vectoriales (donde los datos recopilados o calculados son un vector, como la velocidad).

13.5.1 Visualización de volumen de datos escalares

Para trabajar con datos escalares en tres dimensiones necesita cuatro arreglos tridimensionales:

- datos X, un arreglo tridimensional que contenga la coordenada x de cada punto de retícula.
- datos Y, un arreglo tridimensional que contenga la coordenada y de cada punto de retícula.
- datos Z, un arreglo tridimensional que contenga la coordenada z de cada punto de retícula.
- valores escalares asociados con cada punto de retícula, por ejemplo, una temperatura o presión.

Los arreglos x, y y z usualmente se crean con la función **meshgrid**. Por ejemplo, puede tener

```
x = 1:3;
y = [2,4,6,8];
z = [10, 20];

[X,Y,Z]=meshgrid(x,y,z);
```

Los cálculos producen tres arreglos que son $4 \times 3 \times 2$ y definen la ubicación de cada punto de retícula. El cuarto arreglo requerido es del mismo tamaño y contiene los datos medidos o los valores calculados. MATLAB incluye muchos archivos de datos internos que contienen este tipo de datos, por ejemplo

- datos MRI (almacenados en un archivo llamado MRI).
- datos de campo de flujo (calculados desde un archivo-m).

La función **help** contiene numerosos ejemplos de enfoques de visualización que usan estos datos. Las gráficas que se muestran en la figura 13.29 son una rebanada de contorno de los datos MRI y una isosuperficie de los datos de flujo, ambas creadas al seguir los ejemplos en el tutorial **help**.

Para encontrar estos ejemplos, vaya a la tabla de contenidos del menú de ayuda. Bajo el encabezado MATLAB, encuentre visualización 3-D y luego técnicas de visualización de volumen. Cuando las dos figuras que se muestran se crearon en MATLAB 7.04 para este libro, fue necesario limpiar la figura (**clf**) cada vez antes de representar las imágenes, un detalle no anotado en el tutorial. Cuando no se usó el comando **clf**, las gráficas se comportaron como

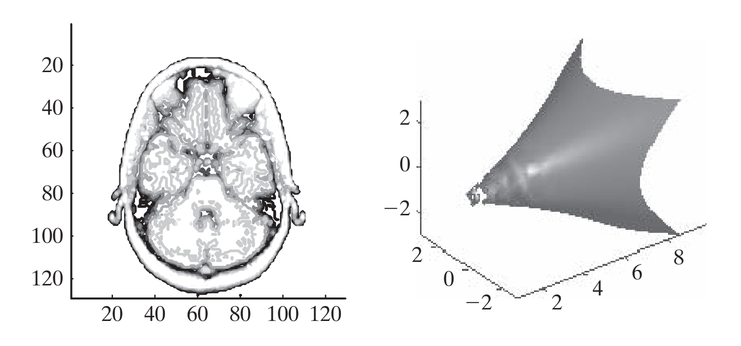


Figura 13.29
MATLAB incluye técnicas de visualización usadas con datos tridimensionales. Izquierda: rebanada de contorno de datos MRI, a partir del archivo de datos de muestra incluido con MATLAB. Derecha: isosuperficie de datos de flujo, a partir del archivo-m de muestra incluido en MATLAB.

si estuviera activado el comando **hold on**. Ésta es una idiosincrasia que se puede corregir en versiones posteriores.

13.5.2 Visualización de volumen de datos vectoriales

Para desplegar datos vectoriales necesita seis arreglos tridimensionales:

- tres arreglos para definir las posiciones x, y y z de cada punto de retícula.
- tres arreglos para definir los datos vectoriales u, v y w.

Un conjunto de muestra de datos de volumen vectoriales llamado **wind** se incluye en MATLAB como archivo de datos. El comando

```
load wind
```

envía seis arreglos tridimensionales al área de trabajo. La visualización de este tipo de datos se puede lograr con varias técnicas diferentes, como son

- gráficas de conos.
- líneas de corriente.
- gráficas de rotacional.

De manera alternativa, los datos vectoriales se pueden procesar en datos escalares, y se pueden usar las técnicas utilizadas en la sección anterior. Por ejemplo, las velocidades no son sólo rapideces; son rapideces más información de dirección. Por tanto, las velocidades son datos vectoriales, con componentes (llamados u, v y w, respectivamente) en las direcciones x, y y z. Se podría convertir velocidades a rapideces con la fórmula

velocidad: rapidez más información de dirección

```
speed = sqrt(u.^2 + v.^2 + w.^2)
```

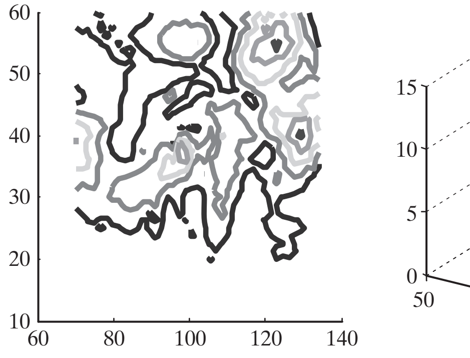
Los datos de rapidez se podrían representar como una o más rebanadas de contorno o como isosuperficies (entre otras técnicas). La imagen izquierda de la figura 13.30 es la gráfica **contourslice** de la rapidez en el conjunto de datos a la octava elevación (z), producido por

```
contourslice(x,y,z,speed,[ ],[ ],8)
```

y la imagen derecha es un conjunto de rebanadas de contorno. La gráfica se ajustó interactivamente de modo que pudieran ver las cuatro rebanadas.

```
contourslice(x,y,z,speed,[ ],[ ],[1,5,10,15])
```

Una gráfica de conos de los mismos datos probablemente sea más reveladora. Siga el ejemplo que se usa en la descripción de la función **coneplot**, en el tutorial **help**, para crear la gráfica de conos que se muestra en la figura 13.31.



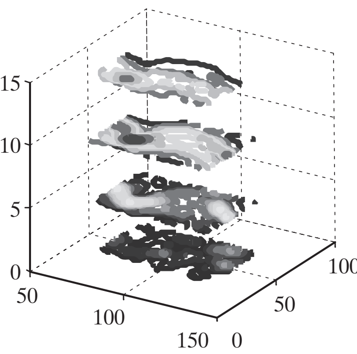


Figura 13.30
Rebanadas de contorno de los datos de rapidez de viento que se incluyen con el programa MATLAB.

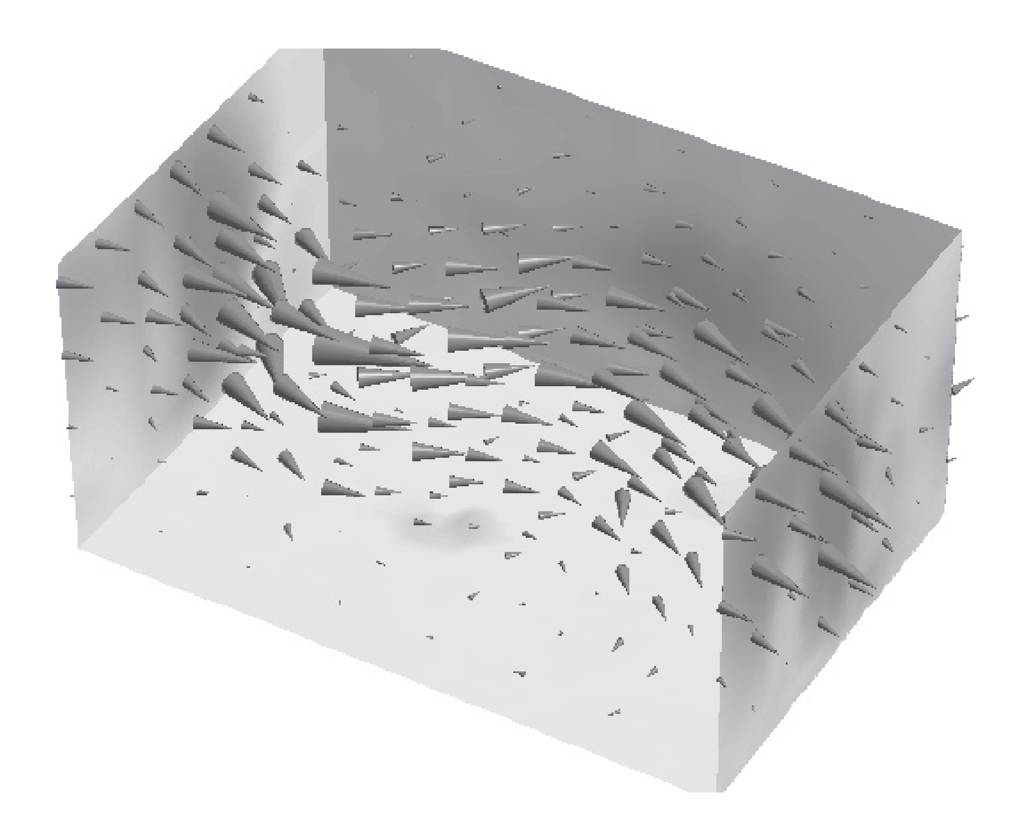


Figura 13.31
Gráfica de conos de los datos de velocidad de viento incluidos con el programa MATLAB.

RESUMEN

MATLAB reconoce tres diferentes técnicas para almacenar y representar imágenes:

Imágenes de intensidad (escala de grises)
Imágenes indexadas
Imágenes RGB (o color verdadero)

La función **imagesc** se usa para desplegar ***imágenes de intensidad***, que a veces se llaman imágenes en escala de grises. Las ***imágenes indexadas*** se despliegan con la función **image** y requieren un mapa de color para determinar la coloración adecuada de la imagen. Se puede crear un mapa de color personalizado para cada imagen, o se puede usar un mapa de color interno. Las imágenes **RGB** (***color verdadero***) también se despliegan con la función **image**, pero no requieren un mapa de color, pues la información de color se incluye en el archivo de imagen.

Si no sabe con qué tipo de datos de imagen trata, puede usar la función **imfinfo** para analizar el archivo. Una vez que sepa qué tipo de archivo tiene, la función **imread** puede cargar un archivo de imagen en MATLAB, o puede usar los controles interactivos de datos del software. El comando **load** puede cargar un archivo **.dat** o uno **.mat**. Para guardar una imagen en uno de los formatos de imagen estándar, use la función **imwrite** o los controles interactivos de datos. También puede guardar los datos de imagen como archivos **.dat** o **.mat**, mediante el comando **save**.

Un handle es un apodo dado a un objeto en MATLAB. Las gráficas que se despliegan en MATLAB incluyen varios objetos diferentes, a todos los cuales se les puede dar un handle. El objeto gráfico fundamental es la figura. En capas arriba de la figura está el objeto eje, y en capas arriba del mismo está el objeto de gráfica real. Cada uno de estos objetos incluye propiedades que se pueden determinar con la función **get** o cambiar con la función **set**. Si

no conoce el nombre de handle apropiado, la función **gcf** (get current figure: obtener figura actual) regresa el handle de la figura actual y **gca** (get current axis: obtener eje actual) regresa el handle de eje actual. La función **set** se usa para cambiar las propiedades de un objeto MATLAB. Por ejemplo, para cambiar el color de una gráfica (la línea que dibujó) llamada **h**, use

```
set(h,'color','red')
```

La animación en MATLAB se manipula con una de dos técnicas: volver a dibujar y borrar, o crear una película. Por lo general, volver a dibujar y borrar es más sencillo para animaciones que representan datos que se pueden calcular rápidamente y no son visualmente complicados. Para tareas que toman significativa potencia computacional, por lo general es más sencillo capturar cuadros individuales y luego combinarlos en una película para verlos tiempo después.

Las superficies complejas con frecuencia son difíciles de visualizar, en especial cuando pueda haber superficies bajo otras superficies. Es posible representar estas superficies ocultas con una transparencia especificada, que le permite ver los detalles oscurecidos. Esto se logra con la función **alpha**. La entrada a esta función puede variar entre 0 y 1, que varía desde completamente transparente a opaco.

Para hacer las superficies más fáciles de interpretar, por defecto las líneas ocultas no se dibujan. El comando **hidden off** fuerza al programa a dibujar dichas líneas.

Aunque MATLAB incluye una extensa capacidad de manipulación de iluminación, usualmente es suficiente encender o apagar la función de iluminación directa. Por defecto, la iluminación es difusa, pero se puede cambiar para dirigir con la función **camlight**.

Las técnicas de visualización de volumen le permiten desplegar datos tridimensionales en varias formas diferentes. Los datos de volumen caen en dos categorías: datos escalares y datos vectoriales. Los datos escalares involucran propiedades como temperatura o presión, y los datos vectoriales incluyen propiedades como velocidades o fuerzas. La función **help** de MATLAB contiene varios ejemplos de técnicas de visualización.

RESUMEN MATLAB

El siguiente resumen MATLAB menciona y describe brevemente todos los caracteres, comandos y funciones especiales que se definieron en este capítulo:

Comandos y funciones	
`alpha`	establece la transparencia del objeto de gráfica actual
`axis`	controla las propiedades del eje de figura
`bone`	mapa de color que hace que una imagen parezca radiografía
`cape`	archivo de imagen MATLAB de muestra de una gorra ¿capa?
`camlight`	enciende la luz de cámara
`clown`	archivo de imagen MATLAB de muestra de un payaso
`colormap`	define cuál mapa de color deben usar las funciones de graficación
`coneplot`	crea una gráfica con marcadores que indican la dirección de los vectores de entrada
`contourslice`	crea una gráfica de contorno a partir de una rebanada de datos
`detail`	archivo de imagen MATLAB de muestra de una sección de un grabado de Durero
`drawnow`	fuerza a MATLAB a dibujar una gráfica inmediatamente
`durer`	archivo de imagen MATLAB de muestra de un grabado de Durero
`earth`	archivo de imagen MATLAB de muestra de la Tierra
`flujet`	archivo de imagen MATLAB de muestra que presenta el comportamiento de un fluido
`gatlin`	archivo de imagen MATLAB de muestra de una fotografía
`gca`	obtiene handle de eje actual
`gcf`	obtiene handle de figura actual

(*Continúa*)

Comandos y funciones (continuación)	
`get`	regresa las propiedades de un objeto especificado
`getframe`	obtiene la figura actual y la guarda como cuadro de película en un arreglo estructura
`gray`	mapa de color que se usa para imágenes en escala de grises
`hidden off`	fuerza a MATLAB a desplegar líneas de retícula oscurecidas
`image`	crea una imagen bidimensional
`imagesc`	crea una imagen bidimensional al escalar los datos
`imfinfo`	lee un archivo gráfico estándar y determina qué tipo de datos contiene
`imread`	lee un archivo gráfico
`imwrite`	escribe un archivo gráfico
`isosurface`	crea una superficie que conecta datos de volumen, todos de la misma magnitud
`mandrill`	archivo de imagen MATLAB de muestra de un mandril
`movie`	reproduce una película almacenada como arreglo de estructura MATLAB
`mri`	conjunto de datos MRI de muestra
`pcolor`	gráfica en seudocolor (similar a una gráfica de contorno)
`peaks`	crea una gráfica de muestra
`set`	establece las propiedades asignadas a un objeto especificado
`shading`	determina la técnica de sombreado que se usa en gráficas de superficie y gráficas en seudocolor
`spine`	archivo de imagen MATLAB de muestra de la radiografía de una columna vertebral
`wind`	archivo de datos MATLAB de muestra de información de velocidad de viento

TÉRMINOS CLAVE

datos escalares
datos vectoriales
gráfica de imagen
gráfica de superficie
handle
imagen de intensidad
imagen indexada
objeto
RGB (color verdadero)
visualización de volumen

PROBLEMAS

13.1 En Internet, encuentre un ejemplo de una imagen de intensidad, una imagen indexada y una imagen RGB. Importe estas imágenes a MATLAB y despliéguelas como figuras MATLAB.

13.2 Un conjunto Julia cuadrático tiene la forma

$$z(n + 1) = z(n)^2 + c$$

El caso especial donde $c = -0.123 + 0.745i$ se llama fractal conejo de Douday. Siga el ejemplo 13.1 y cree una imagen usando este valor de c. Para la imagen Mandelbrot, comience con todos los valores z iguales a 0. Necesitará comenzar con $z = x + yi$. Haga que x y y varíen desde -1.5 hasta 1.5.

13.3 Un conjunto Julia cuadrático tiene la forma

$$z(n + 1) = z(n)^2 + c$$

El caso especial donde $c = -0.391 - 0.587i$ se llama fractal disco Siegel. Siga el ejemplo 13.1 y cree una imagen usando este valor de c. Para la imagen Mandelbrot, comience con todos los valores z iguales a 0. Necesitará comenzar con $z = x + yi$. Haga que x y y varíen desde -1.5 hasta 1.5.

13.4 Un conjunto Julia cuadrático tiene la forma

$$z(n + 1) = z(n)^2 + c$$

El caso especial donde $c = -0.75$ se llama fractal san Marco. Siga el ejemplo 13.1 y cree una imagen usando este valor de c. Para la imagen Mandelbrot, comience con todos los valores z iguales a 0. Necesitará comenzar con $z = x + yi$. Haga que x y y varíen desde -1.5 hasta 1.5.

13.5 Cree una gráfica de la función

$$y = \text{sen}(x) \qquad \text{para } x \text{ desde } -2\pi \text{ hasta } +2\pi$$

Asigne un handle a la gráfica y use la función **set** para cambiar las siguientes propiedades (si no está seguro de cuál es el nombre del objeto para una propiedad dada, use la función **get** para ver una lista de nombres de propiedad disponibles):

(a) color de línea de azul a verde
(b) estilo de línea a rayado
(c) ancho de línea a 2

13.6 Asigne un handle a la figura creada en el problema 13.5 y use la función **set** para cambiar las siguientes propiedades (si no está seguro de cuál es el nombre del objeto para una propiedad dada, use la función **get** para ver una lista de nombres de propiedad disponibles):

(a) color de fondo de figura a rojo
(b) nombre de figura a "Una función seno"

13.7 Asigne un handle a los ejes creados en el problema 13.5 y use la función **set** para cambiar las siguientes propiedades (si no está seguro de cuál es el nombre del objeto para una propiedad dada, use la función **get** para ver una lista de nombres de propiedad disponibles):

(a) color de fondo a azul
(b) escala de eje x a log

13.8 Repita los tres problemas anteriores y cambie las propiedades mediante el inspector interactivo de propiedades. Experimente con otras propiedades y observe los resultados en sus gráficas.

13.9 Cree una animación de la función

$$y = \text{sen}(x - a) \qquad \text{para} \qquad \begin{array}{l} x \text{ que varía de } -2\pi \text{ a } +2\pi \\ a \text{ que varía de 0 a } 8\pi \end{array}$$

- Use un tamaño de paso para x que resulta en una gráfica suave.
- Sea a la variable de animación. (Dibuje una nueva imagen para cada valor de a.)
- Use un tamaño de paso para a que cree una animación suave. Un tamaño de paso muy pequeño hará que la animación parezca moverse más lentamente.

13.10 Cree una película de la función descrita en el problema anterior.

13.11 Cree una animación de lo siguiente:

Sea x que varía de -2π a $+2\pi$
Sea $y = \text{sen}(x)$
Sea $z = \text{sen}(x - a)\cos(y - a)$

Sea a la variable de animación.

Recuerde que necesitará hacer malla x y y para crear matrices bidimensionales; use los arreglos resultantes para encontrar z.

13.12 Cree una película de la función descrita en el problema anterior.

13.13 Cree un programa que le permita acercarse al "fractal conejo" descrito en el problema 13.2 y cree una película de los resultados. (Véase el ejemplo 13.2.)

13.14 Use una gráfica de superficie para graficar la función **peaks**. Emita el comando **hold on** y grafique una esfera que encierre toda la gráfica. Ajuste la transparencia de modo que pueda ver el detalle en el interior de la esfera.

13.15 Grafique la función **peaks** y luego emita el comando **camlight**. Experimente con la colocación de **camlight** en diferentes posiciones y observe el efecto sobre su gráfica.

13.16 Cree una gráfica de contorno apilada de los datos MRI, que muestre las capas 1, 8 y 12 de los datos.

13.17 Un ejemplo de visualización MRI se muestra en el tutorial **help**. Copie y pegue los comandos en un archivo-m y corra el ejemplo. Asegúrese de agregar el comando **clf** antes de dibujar cada nueva gráfica.



APÉNDICE

A

Caracteres especiales, comandos y funciones

Las tablas que se presentan en este apéndice están agrupadas por categorías que aproximadamente son paralelas a la organización por capítulos.

Caracteres especiales	Definición matricial	Capítulo
[ ]	forma matrices	Capítulo 2
( )	se usa en enunciados para agrupar operaciones; se usa con un nombre de matriz para identificar elementos específicos	Capítulo 2
,	separa subíndices o elementos de matriz	Capítulo 2
;	separa filas en una definición matricial; suprime la salida cuando se usa en comandos	Capítulo 2
:	se usa para generar matrices; indica todas las filas o todas las columnas	Capítulo 2

Caracteres especiales	Operadores usados en cálculos MATLAB (escalar y arreglo)	Capítulo
=	operador asignación: asigna un valor a una ubicación de memoria; no es lo mismo que una igualdad	Capítulo 2
%	indica un comentario en un archivo-m	Capítulo 2
+	suma escalar y arreglo	Capítulo 2
-	resta escalar y arreglo	Capítulo 2
*	multiplicación escalar y multiplicación en álgebra matricial	Capítulo 2
.*	multiplicación de arreglo (punto multiplicar o punto estrella)	Capítulo 2
/	división escalar y división en álgebra matricial	Capítulo 2
./	división de arreglo (punto dividir o punto diagonal)	Capítulo 2
^	exponenciación escalar y exponenciación matricial en álgebra matricial	Capítulo 2
.^	exponenciación de arreglo (punto potencia o punto carat)	Capítulo 2
...	elipsis: continuación en la línea siguiente	Capítulo 4
[]	matriz vacía	Capítulo 4

Comandos	Formateo	Capítulo
Format +	establece formato sólo a signos más y menos	Capítulo 2
Format compact	establece formato a forma compacta	Capítulo 2
Format long	establece formato a 14 lugares decimales	Capítulo 2
Format long e	establece formato a 14 lugares exponenciales	Capítulo 2
Format loose	establece formato de vuelta a la forma por defecto no compacta	Capítulo 2
Format short	establece formato de vuelta a 4 lugares decimales por defecto	Capítulo 2
Format short e	establece formato a 4 lugares exponenciales	Capítulo 2
Format rat	establece formato a despliegue racional (fraccional)	Capítulo 2

Comandos	Comandos básicos del área de trabajo	Capítulo
`ans`	nombre de variable por defecto para resultados de cálculos MATLAB	Capítulo 2
`clc`	limpia la pantalla de comando	Capítulo 2
`clear`	limpia el área de trabajo	Capítulo 2
`exit`	termina MATLAB	Capítulo 2
`help`	invoca la utilidad de ayuda	Capítulo 2
`load`	carga matrices desde un archivo	Capítulo 2
`quit`	termina MATLAB	Capítulo 2
`save`	guarda variables en un archivo	Capítulo 2
`who`	menciona las variables en memoria	Capítulo 2
`whos`	menciona las variables y sus tamaños	Capítulo 2
`help`	abre la función ayuda	Capítulo 3
`helpwin`	abre la función de ayuda en ventana	Capítulo 3
`clock`	regresa la hora	Capítulo 3
`date`	regresa la fecha	Capítulo 3
`intmax`	regresa el número entero más grande posible que se usa en MATLAB	Capítulo 3
`intmin`	regresa el número entero más pequeño posible que se usa en MATLAB	Capítulo 3
`realmax`	regresa el número punto flotante más grande posible que se usa en MATLAB	Capítulo 3
`realmin`	regresa el número punto flotante más pequeño posible que se usa en MATLAB	Capítulo 3
`ascii`	indica que los datos se deben guardar en un formato estándar ASCII	Capítulo 2
`pause`	pausa en la ejecución de un programa hasta oprimir cualquier tecla	Capítulo 5

Funciones especiales	Funciones con significado especial que no requiere una entrada	Capítulo
`pi`	aproximación numérica del valor de π	Capítulo 2
`eps`	diferencia más pequeña reconocida	Capítulo 3
`i`	número imaginario	Capítulo 3
`Inf`	infinito	Capítulo 3
`j`	número imaginario	Capítulo 3
`NaN`	no es un número	Capítulo 3

Funciones	Matemática elemental	Capítulo
`abs`	calcula el valor absoluto de un número real o la magnitud de un número complejo	Capítulo 3
`erf`	calcula la función error	Capítulo 3
`exp`	calcula el valor de e^x	Capítulo 3
`factor`	encuentra los factores primos	Capítulo 3
`factorial`	calcula el factorial	Capítulo 3
`gcd`	encuentra el máximo común denominador	Capítulo 3
`isprime`	determina si un valor es primo	Capítulo 3
`isreal`	determina si un valor es real o complejo	Capítulo 3
`lcn`	encuentra el mínimo común denominador	Capítulo 3
`log`	calcula el logaritmo natural, o logaritmo base *e* ($\log_e$)	Capítulo 3
`log10`	calcula el logaritmo común, o logaritmo base 10 ($\log_{10}$)	Capítulo 3
`log2`	calcula el logaritmo base 2 ($\log_2$)	Capítulo 3
`nthroot`	encuentra la *n*-ésima raíz real de la matriz de entrada	Capítulo 3
`primes`	encuentra los números primos menores que el valor de entrada	Capítulo 3
`prod`	multiplica los valores en un arreglo	Capítulo 3
`rats`	convierte la entrada a una representación racional (es decir, una fracción)	Capítulo 3
`rem`	calcula el resto en un problema de división	Capítulo 3
`sign`	determina el signo (positivo o negativo)	Capítulo 3
`sqrt`	calcula la raíz cuadrada de un número	Capítulo 3
`sum`	suma los valores en un arreglo	Capítulo 3

Funciones	Trigonometría	Capítulo
`asin`	calcula el seno inverso (arcoseno)	Capítulo 3
`asind`	calcula el seno inverso y reporta el resultado en grados	Capítulo 3
`cos`	calcula el coseno	Capítulo 3
`sin`	calcula el seno, con radianes como entrada	Capítulo 3
`sind`	calcula el seno, con ángulos en grados como entrada	Capítulo 3
`sinh`	calcula el seno hiperbólico	Capítulo 3
`tan`	calcula la tangente, con radianes como entrada	Capítulo 3

MATLAB incluye todas las funciones trigonométricas; aquí sólo se incluyen las que se discutieron específicamente en el texto.

Funciones	Números complejos	Capítulo
abs	calcula el valor absoluto de un número real o la magnitud de un número complejo	Capítulo 3
angle	calcula el ángulo cuando los números complejos se representan con coordenadas polares	Capítulo 3
complex	crea un número complejo	Capítulo 3
conj	crea la conjugada compleja de un número complejo	Capítulo 3
imag	extrae el componente imaginario de un número complejo	Capítulo 3
isreal	determina si un valor es real o complejo	Capítulo 3
real	extrae el componente real de un número complejo	Capítulo 3

Funciones	Redondeo	Capítulo
ceil	redondea al entero más cercano hacia infinito positivo	Capítulo 3
fix	redondea al entero más cercano hacia cero	Capítulo 3
floor	redondea al entero más cercano hacia menos infinito	Capítulo 3
round	redondea al entero más cercano	Capítulo 3

Funciones	Análisis de datos	Capítulo
cumprod	calcula el producto acumulado de los valores en un arreglo	Capítulo 3
cumsun	calcula la suma acumulada de los valores en un arreglo	Capítulo 3
length	determina la dimensión más grande de un arreglo	Capítulo 3
max	encuentra el valor máximo en un arreglo y determina cuál elemento almacena el valor máximo	Capítulo 3
mean	calcula el promedio de los elementos en un arreglo	Capítulo 3
median	encuentra la mediana de los elementos en un arreglo	Capítulo 3
min	encuentra el valor mínimo en un arreglo y determina cuál elemento almacena el valor mínimo	Capítulo 3
size	determina el número de filas y columnas en un arreglo	Capítulo 3
sort	ordena los elementos de un vector	Capítulo 3
sortrows	ordena las filas de un vector sobre la base de los valores en la primera columna	Capítulo 3
prod	multiplica los valores en un arreglo	Capítulo 3
sum	suma los valores en un arreglo	Capítulo 3
std	determina la desviación estándar	Capítulo 3
var	calcula la varianza	Capítulo 3

Funciones	Números aleatorios	Capítulo
rand	calcula números aleatorios distribuidos uniformemente	Capítulo 3
randn	calcula números aleatorios distribuidos normalmente (gaussianos)	Capítulo 3

Funciones	Formulación, manipulación y análisis matricial	Capítulo
meshgrid	mapea vectores en un arreglo bidimensional	Capítulos 4 y 5
diag	extrae la diagonal de una matriz	Capítulo 4
fliplr	voltea una matriz en su imagen especular de izquierda a derecha	Capítulo 4
flipud	voltea una matriz verticalmente	Capítulo 4
linspace	función vector espaciado linealmente	Capítulo 2
logspace	función vector espaciada logarítmicamente	Capítulo 2
cross	calcula el producto cruz	Capítulo 9
det	calcula el determinante de una matriz	Capítulo 9
dot	calcula el producto punto	Capítulo 9
inv	calcula el inverso de una matriz	Capítulo 9

Funciones	Gráficas bidimensionales	Capítulo
bar	genera una gráfica de barras	Capítulo 5
barh	genera una gráfica de barras horizontal	Capítulo 5
contour	genera un mapa de contorno de una superficie tridimensional	Capítulo 5
comet	dibuja una gráfica *x-y* en una secuencia de falsa animación	Capítulo 5
fplot	crea una gráfica *x-y* con base en una función	Capítulo 5
hist	genera un histograma	Capítulo 5
loglog	genera una gráfica *x-y* con ambos ejes en escala logarítmica	Capítulo 5
pcolor	crea una gráfica en seudocolor similar a un mapa de contorno	Capítulo 5
pie	genera una gráfica de pastel	Capítulo 5
plot	crea una gráfica *x-y*	Capítulo 5
plotyy	crea una gráfica con dos ejes *y*	Capítulo 5
polar	crea una gráfica polar	Capítulo 5
semilogx	genera una gráfica *x-y* con el eje *x* en escala logarítmica	Capítulo 5
semilogy	genera una gráfica *x-y* con el eje *y* en escala logarítmica	Capítulo 5

Funciones	Gráficas tridimensionales	Capítulo
`bar3`	genera una gráfica de barras tridimensional	Capítulo 5
`bar3h`	genera una gráfica de barras tridimensional horizontal	Capítulo 5
`comet3`	dibuja una gráfica de línea tridimensional en una secuencia de falsa animación	Capítulo 5
`mesh`	genera una gráfica de malla de una superficie	Capítulo 5
`peaks`	crea una matriz tridimensional de muestra que se usa para demostrar las funciones de graficación	Capítulo 5
`pie3`	genera una gráfica de pastel tridimensional	Capítulo 5
`plot3`	genera una gráfica de línea tridimensional	Capítulo 5
`sphere`	función de muestra que se usa para demostrar la graficación	Capítulo 5
`surf`	genera una gráfica de superficie	Capítulo 5
`surfc`	genera una combinación de gráfica de superficie y contorno	Capítulo 5

Caracteres especiales	Control de apariencia de gráfica	Capítulo
Indicador	**Tipo de línea**	
-	sólida	Capítulo 5
:	punteada	Capítulo 5
-.	raya-punto	Capítulo 5
--	rayada	Capítulo 5
Indicador	**Tipo de punto**	
.	punto	Capítulo 5
o	círculo	Capítulo 5
x	marca x	Capítulo 5
+	más	Capítulo 5
*	estrella	Capítulo 5
s	cuadrado	Capítulo 5
d	diamante	Capítulo 5
v	triángulo abajo	Capítulo 5
^	triángulo arriba	Capítulo 5
<	triángulo izquierdo	Capítulo 5
>	triángulo derecho	Capítulo 5
p	pentagrama	Capítulo 5
h	hexagrama	Capítulo 5
Indicador	**Color**	
b	azul	Capítulo 5
g	verde	Capítulo 5
r	rojo	Capítulo 5
c	cian	Capítulo 5
m	magenta	Capítulo 5
y	amarillo	Capítulo 5
k	negro	Capítulo 5

Funciones	Control y anotación de figura	Capítulo
axis	congela el escalamiento del eje actual para gráficas posteriores o especifica las dimensiones del eje	Capítulo 5
axis equal	fuerza el mismo espaciamiento de escala para cada eje	Capítulo 5
colormap	esquema de color usado en gráficas de superficie	Capítulo 5
figure	abre una nueva ventana de figura	Capítulo 5
grid	agrega una retícula sólo a la gráfica actual	Capítulo 5
grid off	desactiva la retícula	Capítulo 5
grid on	agrega una retícula a las gráficas actual y todas las subsecuentes en la figura actual	Capítulo 5
hold off	instruye a MATLAB a borrar los contenidos de figura antes de agregar nueva información	Capítulo 5
hold on	instruye a MATLAB a no borrar los contenidos de figura antes de agregar nueva información	Capítulo 5
legend	agrega una leyenda a una gráfica	Capítulo 5
shading flat	sombrea una gráfica de superficie con un color por sección de retícula	Capítulo 5
shading interp	sombrea una gráfica de superficie mediante interpolación	Capítulo 5
subplot	divide la ventana de gráficas en secciones disponibles para graficación	Capítulo 5
text	agrega un recuadro de texto a una gráfica	Capítulo 5
title	agrega un título a una gráfica	Capítulo 5
xlabel	agrega una etiqueta al eje *x*	Capítulo 5
ylabel	agrega una etiqueta al eje *y*	Capítulo 5
zlabel	agrega una etiqueta al eje *z*	Capítulo 5

Funciones	Esquemas de color de figura	Capítulo
autumn	mapa de color opcional usado en gráficas de superficie	Capítulo 5
bone	mapa de color opcional usado en gráficas de superficie	Capítulo 5
colorcube	mapa de color opcional usado en gráficas de superficie	Capítulo 5
cool	mapa de color opcional usado en gráficas de superficie	Capítulo 5
copper	mapa de color opcional usado en gráficas de superficie	Capítulo 5
flag	mapa de color opcional usado en gráficas de superficie	Capítulo 5
hot	mapa de color opcional usado en gráficas de superficie	Capítulo 5
hsv	mapa de color opcional usado en gráficas de superficie	Capítulo 5
jet	mapa de color por defecto usado en gráficas de superficie	Capítulo 5
pink	mapa de color opcional usado en gráficas de superficie	Capítulo 5
prism	mapa de color opcional usado en gráficas de superficie	Capítulo 5
spring	mapa de color opcional usado en gráficas de superficie	Capítulo 5
summer	mapa de color opcional usado en gráficas de superficie	Capítulo 5
white	mapa de color opcional usado en gráficas de superficie	Capítulo 5
winter	mapa de color opcional usado en gráficas de superficie	Capítulo 5

Funciones y caracteres especiales	Creación y uso de función	Capítulo
addpath	agrega un directorio a la ruta de búsqueda de MATLAB	Capítulo 6
function	identifica un archivo-m como función	Capítulo 6
nargin	determina el número de argumentos de entrada en una función	Capítulo 6
nargout	determina el número de argumentos de salida de una función	Capítulo 6
pathtool	abre la herramienta de ruta interactiva	Capítulo 6
varargin	indica que un número variable de argumentos puede ser entrada a una función	Capítulo 6
@	identifica un manipulador de función, como cualquiera de los usados con las funciones en línea	Capítulo 6
%	comentario	Capítulo 6

Caracteres especiales	Control de formato	Capítulo
'	comienza y termina una cadena	Capítulo 7
%	marcador de posición (placeholder) usado en el comando **fprintf**	Capítulo 7
%f	notación punto fijo o decimal	Capítulo 7
%e	notación exponencial	Capítulo 7
%g	notación o punto fijo o exponencial	Capítulo 7
%s	notación cadena	Capítulo 7
%%	divisor de celda	Capítulo 7
\n	salto de línea (linefeed)	Capítulo 7
\r	regreso de carro (similar a linefeed)	Capítulo 7
\t	tabulador	Capítulo 7
\b	retroceder un espacio (backspace)	Capítulo 7

Funciones	Control entrada/salida (I/O)	Capítulo
disp	despliega una cadena o una matriz en la ventana de comandos	Capítulo 7
fprintf	controla el despliegue de la ventana de comandos	Capítulo 7
ginput	permite al usuario elegir valores de una gráfica	Capítulo 7
input	permite al usuario ingresar valores	Capítulo 7
pause	pausa el programa	Capítulo 7
uiimport	lanza el Asistente de Importación	Capítulo 7
wavread	lee archivos wave	Capítulo 7
xlsimport	importa archivos de datos Excel	Capítulo 7
xlswrite	exporta datos como un archivo Excel	Capítulo 7
load	carga matrices desde un archivo	Capítulo 2
save	guarda variables en un archivo	Capítulo 2
celldisp	despliega los contenidos de un arreglo celda	Capítulo 10
imfinfo	lee un archivo gráfico estándar y determina qué tipo de datos contiene	Capítulo 13
imread	lee un archivo de gráficos	Capítulo 13
imwrite	escribe un archivo de gráficos	Capítulo 13

Funciones	Operadores de comparación	Capítulo
<	menor que	Capítulo 8
<=	menor que o igual a	Capítulo 8
>	mayor que	Capítulo 8
>=	mayor que o igual a	Capítulo 8
==	igual a	Capítulo 8
~=	no igual a	Capítulo 8

<table>
<tr><th>Caracteres especiales</th><th>Operadores lógicos</th><th>Capítulo</th></tr>
<tr><td>&</td><td>and</td><td>Capítulo 8</td></tr>
<tr><td>|</td><td>or</td><td>Capítulo 8</td></tr>
<tr><td>~</td><td>not</td><td>Capítulo 8</td></tr>
<tr><td>xor</td><td>or exclusiva</td><td>Capítulo 8</td></tr>
</table>

Funciones	Estructuras de control	Capítulo
`break`	termina la ejecución de un bucle	Capítulo 8
`case`	ordena respuestas	Capítulo 8
`continue`	termina el paso actual a través de un bucle, pero procede al siguiente paso	Capítulo 8
`else`	define la ruta si el resultado de un enunciado **if** es falso	Capítulo 8
`elseif`	define la ruta si el resultado de un enunciado **if** es falso y especifica una nueva prueba lógica	Capítulo 8
`end`	identifica el final de una estructura de control	Capítulo 8
`for`	genera una estructura bucle (loop)	Capítulo 8
`if`	verifica una condición que resulta en verdadero o en falso	Capítulo 8
`menu`	crea un menú a usar como vehículo de entrada	Capítulo 8
`otherwise`	parte de la estructura de selección de caso	Capítulo 8
`switch`	parte de la estructura de selección de caso	Capítulo 8
`while`	genera una estructura bucle	Capítulo 8

Funciones	Funciones lógicas	Capítulo
`all`	verifica si un criterio se satisface por todos los elementos en un arreglo	Capítulo 8
`any`	verifica si un criterio se satisface por alguno de los elementos en un arreglo	Capítulo 8
`find`	determina cuáles elementos en una matriz satisfacen el criterio de entrada	Capítulo 8
`isprime`	determina si un valor es primo	Capítulo 3
`isreal`	determina si un valor es real o complejo	Capítulo 3

Funciones	Cronometrado	Capítulo
`clock`	determina el tiempo actual en el reloj del CPU	Capítulo 8
`etime`	encuentra el tiempo transcurrido	Capítulo 8
`tic`	comienza una secuencia de cronometrado	Capítulo 8
`toc`	detiene una secuencia de cronometrado	Capítulo 8
`date`	regresa la fecha	Capítulo 3

Funciones	Matrices especiales	Capítulo
`eye`	genera una matriz identidad	Capítulo 9
`magic`	crea una matriz “mágica”	Capítulo 9
`ones`	crea una matriz que contiene todos unos	Capítulo 9
`pascal`	crea una matriz de Pascal	Capítulo 9
`zeros`	crea una matriz que contiene todos ceros	Capítulo 9
`gallery`	contiene matrices ejemplo	Capítulo 9

Caracteres especiales	Tipos de datos	Capítulo
{ }	constructor de arreglo celda	Capítulos 10 y 11
''	datos cadena (información carácter)	Capítulos 10 y 11
abc	arreglo carácter	Capítulo 10
	arreglo numérico	Capítulo 10
	arreglo simbólico	Capítulo 10
	arreglo lógico	Capítulo 10
	arreglo esparcido	Capítulo 10
{ }	arreglo celda	Capítulo 10
	arreglo estructura	Capítulo 10

Funciones	Manipulación de tipo de datos	Capítulo
`celldisp`	despliega los contenidos de un arreglo celda	Capítulo 10
`char`	crea un arreglo carácter acolchado	Capítulo 10
`double`	cambia un arreglo a un arreglo de doble precisión	Capítulo 10
`int16`	entero signado de 16 bits	Capítulo 10
`int32`	entero signado de 32 bits	Capítulo 10
`int64`	entero signado de 64 bits	Capítulo 10
`int8`	entero signado de 8 bits	Capítulo 10
`num2str`	convierte un arreglo numérico a un arreglo carácter	Capítulo 10
`single`	cambia un arreglo a un arreglo de precisión sencillo	Capítulo 10
`sparse`	convierte una matriz de formato completo a una matriz de formato esparcido	Capítulo 10
`str2num`	convierte un arreglo carácter a un arreglo numérico	Capítulo 10
`uint16`	entero no signado de 16 bits	Capítulo 10
`uint32`	entero no signado de 32 bits	Capítulo 10
`uint64`	entero no signado de 64 bits	Capítulo 10
`uint8`	entero no signado de 8 bits	Capítulo 10

Funciones	Manipulación de expresiones simbólicas	Capítulo
collect	recopila términos iguales	Capítulo 11
diff	encuentra la derivada simbólica de una expresión simbólica	Capítulo 11
dsolve	solucionador de ecuación diferencial	Capítulo 11
expand	expande una expresión o ecuación	Capítulo 11
factor	factoriza una expresión o ecuación	Capítulo 11
int	encuentra la integral simbólica de una expresión simbólica	Capítulo 11
numden	extrae el numerador y denominador de una expresión o una ecuación	Capítulo 11
simple	intenta y reporta todas las funciones de simplificación y selecciona la respuesta más corta	Capítulo 11
simplify	simplifica usando las reglas de simplificación internas de Maple	Capítulo 11
solve	resuelve una expresión o ecuación simbólica	Capítulo 11
subs	sustituye en una expresión o ecuación simbólica	Capítulo 11
sym	crea una variable, expresión o ecuación simbólica	Capítulo 11
syms	crea variables simbólicas	Capítulo 11

Funciones	Graficación simbólica	Capítulo
ezcontour	crea una gráfica de contorno	Capítulo 11
ezcontourf	crea una gráfica de contorno llena	Capítulo 11
ezmesh	crea una gráfica de malla a partir de una expresión simbólica	Capítulo 11
ezmeshc	grafica tanto una gráfica de malla como una de contorno creada a partir de una expresión simbólica	Capítulo 11
ezplot	crea una gráfica x-y de una expresión simbólica	Capítulo 11
ezplot3	crea una gráfica de línea tridimensional	Capítulo 11
ezpolar	crea una gráfica en coordenadas polares	Capítulo 11
ezsurf	crea una gráfica de superficie a partir de una expresión simbólica	Capítulo 11
ezsurfc	grafica tanto una gráfica de malla como una de contorno creada a partir de una expresión simbólica	Capítulo 11

Funciones	Técnicas numéricas	Capítulo
`cftool`	abre la interfaz gráfica de usuario de ajuste de curva	Capítulo 12
`diff`	calcula las diferencias entre valores adyacentes en un arreglo si la entrada es un arreglo; encuentra la derivada simbólica si la entrada es una expresión simbólica	Capítulo 12
`interp1`	aproxima datos intermedios con la técnica de interpolación lineal por defecto o con un enfoque específico de orden superior	Capítulo 12
`interp2`	función interpolación bidimensional	Capítulo 12
`interp3`	función interpolación tridimensional	Capítulo 12
`interpn`	función interpolación multidimensional	Capítulo 12
`ode45`	solucionador de ecuaciones diferenciales ordinarias	Capítulo 12
`ode23`	solucionador de ecuaciones diferenciales ordinarias	Capítulo 12
`ode113`	solucionador de ecuaciones diferenciales ordinarias	Capítulo 12
`ode15s`	solucionador de ecuaciones diferenciales ordinarias	Capítulo 12
`ode23s`	solucionador de ecuaciones diferenciales ordinarias	Capítulo 12
`ode23t`	solucionador de ecuaciones diferenciales ordinarias	Capítulo 12
`ode23tb`	solucionador de ecuaciones diferenciales ordinarias	Capítulo 12
`ode15i`	solucionador de ecuaciones diferenciales ordinarias	Capítulo 12
`polyfit`	calcula el coeficiente de un polinomio de mínimos cuadrados	Capítulo 12
`polyval`	evalúa un polinomio en un valor específico de *x*	Capítulo 12
`quad`	calcula la integral bajo una curva (Simpson)	Capítulo 12
`quadl`	calcula la integral bajo una curva (Lobatto)	Capítulo 12

Funciones	Conjuntos de datos e imágenes de muestra	Capítulo
`cape`	archivo de imagen MATLAB de muestra de una capa	Capítulo 13
`clown`	archivo de imagen MATLAB de muestra de un payaso	Capítulo 13
`detail`	archivo de imagen MATLAB de muestra de una sección de un grabado en madera de Durero	Capítulo 13
`durer`	archivo de imagen MATLAB de muestra de un grabado de Durero	Capítulo 13
`earth`	archivo de imagen MATLAB de muestra de la Tierra	Capítulo 13
`flujet`	archivo de imagen MATLAB de muestra que presenta comportamiento de fluido	Capítulo 13
`gatlin`	archivo de imagen MATLAB de muestra de una fotografía	Capítulo 13
`mandrill`	archivo de imagen MATLAB de muestra de un mandril	Capítulo 13
`mri`	conjunto de datos IRM de muestra	Capítulo 13
`peaks`	crea una gráfica muestra	Capítulo 13
`spine`	archivo de imagen MATLAB de muestra de una radiografía de columna vertebral	Capítulo 13
`wind`	archivo de datos MATLAB de muestra de información de velocidad de viento	Capítulo 13
`sphere`	función muestra que se usa para demostrar graficación	Capítulo 5
`census`	conjunto de datos interno que se usa para demostrar técnicas numéricas	Capítulo 12
`handel`	conjunto de datos interno que se usa para demostrar la función sound	Capítulo 3

Funciones	Visualización avanzada	Capítulo
`alpha`	establece la transparencia del objeto de gráfica actual	Capítulo 13
`camlight`	enciende la luz de cámara	Capítulo 13
`coneplot`	crea una gráfica con marcadores que indican la dirección de los vectores de entrada	Capítulo 13
`contourslice`	crea una gráfica de contorno a partir de una rebanada de datos	Capítulo 13
`drawnow`	fuerza a MATLAB a dibujar una gráfica inmediatamente	Capítulo 13
`gca`	obtiene manipulador de eje actual	Capítulo 13
`gcf`	obtiene manipulador de figura actual	Capítulo 13
`get`	regresa las propiedades de un objeto específico	Capítulo 13
`getframe`	obtiene la figura actual y la guarda como un cuadro de película en un arreglo estructura	Capítulo 13
`image`	crea una imagen bidimensional	Capítulo 13
`imagesc`	crea una imagen bidimensional al escalar los datos	Capítulo 13
`imfinfo`	lee un archivo gráfico estándar y determina qué tipo de datos contiene	Capítulo 13
`imread`	lee un archivo gráfico	Capítulo 13
`imwrite`	escribe un archivo gráfico	Capítulo 13
`isosurface`	crea superficies que conectan datos volumen de la misma magnitud	Capítulo 13
`movie`	reproduce una película almacenada como un arreglo estructura MATLAB	Capítulo 13
`set`	establece las propiedades asignadas a un objeto específico	Capítulo 13
`shading`	determina la técnica de sombreado que se usa en gráficas de superficie y gráficas de seudocolor	Capítulo 13



APÉNDICE B

Soluciones a ejercicios de práctica

Existen muchas formas para resolver problemas en MATLAB. Estas soluciones representan un abordaje posible.

Ejercicio de práctica 2.1

1. 7
2. 10
3. 2.5000
4. 17
5. 7.8154
6. 4.1955
7. 12.9600
8. 5
9. 2.2361
10. −1

Ejercicio de práctica 2.2

1. **test** es un nombre válido.
2. **Test** es un nombre válido, pero es una variable diferente de **test**.
3. **if** no se permite. Es una palabra clave reservada.
4. **mi-libro** no se permite porque contiene un guión.
5. **mi_libro** es un nombre válido.
6. **Esteesunnombremuylargoperoinclusoasisepermite?** No se permite porque incluye un signo de interrogación. Incluso sin dicho signo, no es buena idea.
7. **1ergrupo** no se permite porque comienza con un número.
8. **grupo_uno** es un nombre válido.
9. **zzaAbc** es un nombre válido, aunque no es muy bueno porque combina mayúsculas y minúsculas y no es significativo.
10. **z34wAwy%12#** no es válido porque incluye los signos de porcentaje y número.

Ejercicio de práctica 2.3

1. 6
2. 72
3. 16
4. 13
5. 48
6. 38.5
7. 4096
8. 2.4179e + 024
9. 245
10. 2187
11. (5+3)/(9−1) = 1
12. 2^3−4/(5+3) = 7.5
13. 5^(2+1)/(4−1) = 41.6667
14. (4+1/2)*(5+2/3) =25.5
15. (5+6*7/3−2^2)/(2/3*3/(3*6)) = 135

Ejercicio de práctica 2.4

1. `a = [2.3 5.8 9]`
2. `sin(a)`
```
ans =
   0.7457 -0.4646 0.4121
```
3. `a + 3`
```
ans =
    5.3000 8.8000 12.0000
```
4. `b = [5.2 3.14 2]`
5. `a + b`
```
ans =
    7.5000 8.9400 11.0000
```
6. `a .* b`
```
ans =
    11.9600 18.2120 18.0000
```
7. `a.^2`
```
ans =
   5.2900 33.6400 81.0000
```
8. `c = 0:10 or`
   `c = [0:10]`
9. `d = 0:2:10 or`
   `d = [0:2:10]`
10. `linspace(10,20, 6)`
```
ans =
   10 12 14 16 18 20
```
11. `logspace(1, 2, 5)`
```
ans =
   10.0000 17.7828 31.6228 56.2341 100.0000
```

Ejercicio de práctica 3.1

1. En la ventana de comandos, escriba

```
help cos
help sqrt
help exp
```

2. Seleccione **Help** → **MATLAB Help** de la barra de menú.
   Use el panel izquierdo para navegar hacia **Functions – Categorical List** o **Functions – Alphabetical List**
3. Seleccione **Help** → **Web Resources** → **The Mathworks Web Site**

Ejercicio de práctica 3.2

1.
```
x=-2:1:2
x =
  -2 -1 0 1 2
abs(x)
ans =
 2 1 0 1 2
sqrt(x)
ans =
     0 + 1.4142i 0 + 1.0000i 0 1.0000 1.4142
```
2.
```
sqrt(-3)
ans =
  0 + 1.7321i
sqrt(3)
ans =
  1.7321
```
3.
```
x=-10:3:11
x =
  -10 -7 -4 -1 2 5 8 11
x/3
ans =
  -3.3333 -2.3333 -1.3333 -0.3333 0.6667 1.6667 2.6667
   3.6667
rem(x,3)
ans =
    -1 -1 -1 -1 2 2 2 2
```
4.
```
exp(x)
ans =
 1.0e+004 *
0.0000 0.0000 0.0000 0.0000 0.0007 0.0148 0.2981 5.9874
```
5.
```
log(x)
ans =
  Columns 1 through 4
  2.3026 + 3.1416i 1.9459 + 3.1416i 1.3863 + 3.1416i
  0 + 3.1416i
  Columns 5 through 8
  0.6931 1.6094 2.0794 2.3979
log10(x)
ans =
  Columns 1 through 4
  1.0000 + 1.3644i 0.8451 + 1.3644i 0.6021 + 1.3644i
  0 + 1.3644i
  Columns 5 through 8
```
6.
```
sign(x)
ans =
  -1 -1 -1 -1 1 1 1 1
```
7.
```
format rat
x/2
ans =
    -5 -7/2 -2 -1/2 1 5/2 4 11/2
```

Ejercicio de práctica 3.3

1. **factor(322)**
   ```
   ans =
     2 7 23
   ```
2. **gcd(322,6)**
   ```
   ans =
     2
   ```
3. **isprime(322)**
   ```
   ans =
     0
   ```
   Puesto que el resultado de **isprime** es el número 0, 322 no es un número primo.
4. **length(primes(322))**
   ```
   ans =
     66
   ```
5. **rats(pi)**
   ```
   ans =
     355/113
   ```

Ejercicio de práctica 3.4

1. **theta=3*pi;**
   sin(2*theta)
   ```
   ans =
   -7.3479e-016
   ```
2. **theta=0:0.2*pi:2*pi;**
   cos(theta)
   ```
   ans =
    Columns 1 through 7
     1.0000 0.8090 0.3090 -0.3090 -0.8090 -1.0000 -0.8090
    Columns 8 through 11
     -0.3090 0.3090 0.8090 1.0000
   ```
3. **asin(1)**
   ```
   ans =
     1.5708
   ```
   Esta respuesta está en radianes.
4. **acos(x)**
   ```
   ans =
    Columns 1 through 7
     3.1416 2.4981 2.2143 1.9823 1.7722 1.5708 1.3694
    Columns 8 through 11
     1.1593 0.9273 0.6435 0
   ```
5. **cos(45*pi/180)**
   ```
   ans =
     0.7071
   ```
   cosd(45)
   ```
   ans =
     0.7071
   ```

(Continúa)

Ejercicio de práctica 3.4 (*Continuación*)

6. **asin(0.5)**
   ans =
   0.5236 Esta respuesta está en radianes. También podría encontrar el resultado en grados.
   asind(0.5)
   ans =
   30.0000
7. **csc(60*pi/180)**
   ans =
   1.1547 or....

   cscd(60)
   ans =
   1.1547

Ejercicio de práctica 3.5

```
x=[4 90 85 75; 2 55 65 75; 3 78 82 79;1 84 92 93]
x =
    4 90 85 75
    2 55 65 75
    3 78 82 79
    1 84 92 93
```

1. **max(x)**
   ans =
   4 90 92 93
2. **[maximum, row]=max(x)**
   maximum =
   4 90 92 93
   row =
   1 1 4 4
3. **max(x')**
   ans =
   90 75 82 93
4. **[maximum, column]=max(x')**
   maximum =
   90 75 82 93
   column =
   2 4 3 4
5. **max(max(x))**
   ans =
   93

Ejercicio de práctica 3.6

```
x = [4 90 85 75; 2 55 65 75; 3 78 82 79;1 84 92 93];
```

1. `mean(x)`
```
ans =
  2.5000 76.7500 81.0000 80.5000
```
2. `median(x)`
```
ans =
  2.5000 81.0000 83.5000 77.0000
```
3. `mean(x')`
```
ans =
  63.5000 49.2500 60.5000 67.5000
```
4. `median(x')`
```
ans =
80.0000 60.0000 78.5000 88.0000
```

Ejercicio de práctica 3.7

```
x = [4 90 85 75; 2 55 65 75; 3 78 82 79;1 84 92 93];
```

1. `size(x)`
```
ans =
   4 4
```
2. `sort(x)`
```
ans =
   1 55 65 75
   2 78 82 75
   3 84 85 79
   4 90 92 93
```
3. `sort(x,'descend')`
```
ans =
   4 90 92 93
   3 84 85 79
   2 78 82 75
   1 55 65 75
```
4. `sortrows(x)`
```
ans =
   1 84 92 93
   2 55 65 75
   3 78 82 79
   4 90 85 75
```

Ejercicio de práctica 3.8

```
x = [4 90 85 75; 2 55 65 75; 3 78 82 79;1 84 92 93];
```

1. `std(x)`
```
ans =
  1.2910 15.3052 11.4601 8.5440
```
2. `var(x)`
```
ans =
  1.6667 234.2500 131.3333 73.0000
```
3. `sqrt(var(x))`
```
ans =
  1.2910 15.3052 11.4601 8.5440
```
4. La raíz cuadrada de la varianza es igual a la desviación estándar.

Ejercicio de práctica 3.9

1. `rand(3)`
```
ans =
  0.9501 0.4860 0.4565
  0.2311 0.8913 0.0185
  0.6068 0.7621 0.8214
```
2. `randn(3)`
```
ans =
  -0.4326 0.2877 1.1892
  -1.6656 -1.1465 -0.0376
   0.1253 1.1909 0.3273
```
3. `x=rand(100,5);`
4. `max(x)`
```
ans =
  0.9811 0.9785 0.9981 0.9948 0.9962
std(x)
ans =
  0.2821 0.2796 0.3018 0.2997 0.2942
var(x)
ans =
  0.0796 0.0782 0.0911 0.0898 0.0865
mean(x)
ans =
  0.4823 0.5026 0.5401 0.4948 0.5111
```
5. `x=randn(100,5);`
6. `max(x)`
```
ans =
  2.6903 2.6289 2.7316 2.4953 1.7621
 std(x)
ans =
  0.9725 0.9201 0.9603 0.9367 0.9130
 var(x)
ans =
  0.9458 0.8465 0.9221 0.8774 0.8335
 mean(x)
ans =
  -0.0277 0.0117 -0.0822 0.0974 -0.1337
```

Ejercicio de práctica 3.10

1.
```
A=1+i
A =
  1.0000 + 1.0000i
B=2-3i
B =
  2.0000 - 3.0000i
  C=8+2i
  C =
     8.0000 + 2.0000i
```
2.
```
imagD=[-3,8,-16];
realD=[2,4,6];
D=complex(realD,imagD)
ans =
  2.0000 - 3.0000i 4.0000 + 8.0000i 6.0000 -16.0000i
```
3.
```
abs(A)
ans =
  1.4142
abs(B)
ans =
  3.6056
abs(C)
ans =
  8.2462
abs(D)
ans =
  3.6056 8.9443 17.0880
```
4.
```
angle(A)
ans =
  0.7854
angle(B)
ans =
  -0.9828
angle(C)
ans =
  0.2450
angle(D)
ans =
  -0.9828 1.1071 -1.2120
```
5.
```
conj(D)
ans =
  2.0000 + 3.0000i 4.0000 - 8.0000i 6.0000 +16.0000i
```
6.
```
D'
ans =
  2.0000 + 3.0000i
  4.0000 - 8.0000i
  6.0000 +16.0000i
```
7.
```
sqrt(A.*A')
ans =
  1.4142
```

Ejercicio de práctica 3.11

1. `clock`
```
ans =
 1.0e+003 *
  2.0050 0.0070 0.0200 0.0190 0.0440 0.0309
```
2. `date`
```
ans =
20-Jul-2005
```
3. `5*10^500`
```
ans =
 Inf
```
4. `1/5*10^500`
```
ans =
  Inf
```
5. `0/0`
```
Warning: Divide by zero.
ans =
 NaN
```

Ejercicio de práctica 4.1

```
a = [12 17 3 6]
a =
   12 17 3 6
b = [5 8 3; 1 2 3; 2 4 6]
b =
   5 8 3
   1 2 3
   2 4 6
c = [22;17;4]
c =
  22
  17
  4
```

1. `x1 = a(1,2)`
```
x1 =
 17
```
2. `x2 = b(:,3)`
```
x2 =
    3
    3
    6
```
3. `x3 = b(3,:)`
```
x3 =
    2 4 6
```

4. `x4 = [b(1,1), b(2,2), b(3,3)]`

```
x4 =
    5 2 6
```

5. `x5 = [a(1:3);b]`

```
x5 =
   12 17 3
    5  8 3
    1 2 3
    2 4 6
```

6. `x6 = [c,b;a]`

```
x6 =
   22 5 8 3
   17 1 2 3
   4 2 4 6
   12 17 3 6
```

7. `x7=b(8)`

```
x7 =
   3
```

8. `x8=b(:)`

```
x8 =
   5
   1
   2
   8
   2
   4
   3
   3
   6
```

Ejercicio de práctica 4.2

1. `length = [1, 3, 5];`

```
width = [2,4,6,8];
[L,W]=meshgrid(length,width);
area = L.*W
area =
   2  6 10
   4 12 20
   6 18 30
   8 24 40
```

(Continúa)

Ejercicio de práctica 4.2 (*Continuación*)

2.
```
radius = 0:3:12;
height = 10:2:20;
[R,H] = meshgrid(radius,height);
volume = pi*R.^2.*H
volume =
 1.0e+003 *
      0  0.2827  1.1310  2.5447  4.5239
      0  0.3393  1.3572  3.0536  5.4287
      0  0.3958  1.5834  3.5626  6.3335
      0  0.4524  1.8096  4.0715  7.2382
      0  0.5089  2.0358  4.5804  8.1430
      0  0.5655  2.2619  5.0894  9.0478
```

Ejercicio de práctica 4.3

1.
```
zeros(3)
ans =
   0  0  0
   0  0  0
   0  0  0
```
2.
```
zeros(3,4)
ans =
   0  0  0  0
   0  0  0  0
   0  0  0  0
```
3.
```
ones(3)
ans =
   1  1  1
   1  1  1
   1  1  1
```
4.
```
ones(5,3)
ans =
   1  1  1
   1  1  1
   1  1  1
   1  1  1
   1  1  1
```
5.
```
ones(4,6)*pi
ans =
  3.1416  3.1416  3.1416  3.1416  3.1416  3.1416
  3.1416  3.1416  3.1416  3.1416  3.1416  3.1416
  3.1416  3.1416  3.1416  3.1416  3.1416  3.1416
  3.1416  3.1416  3.1416  3.1416  3.1416  3.1416
```

6.
```
x = [1,2,3];
diag(x)
ans =
   1  0  0
   0  2  0
   0  0  3
```

7.
```
x = magic(10)
x =
   92  99   1   8  15  67  74  51  58  40
   98  80   7  14  16  73  55  57  64  41
    4  81  88  20  22  54  56  63  70  47
   85  87  19  21   3  60  62  69  71  28
   86  93  25   2   9  61  68  75  52  34
   17  24  76  83  90  42  49  26  33  65
   23   5  82  89  91  48  30  32  39  66
   79   6  13  95  97  29  31  38  45  72
   10  12  94  96  78  35  37  44  46  53
   11  18 100  77  84  36  43  50  27  59
```

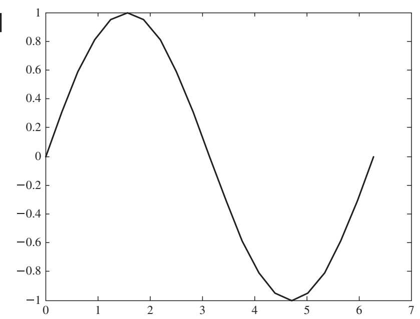
a.
```
diag(x)
ans =
 92   80   88   21   9   42   30   38   46   59
```

b.
```
diag(fliplr(x))
ans =
 40   64   63   62   61   90   89   13   12   11
```

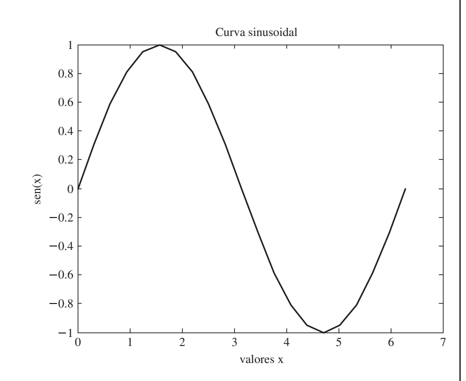
c.
```
sum(x)
ans =
 505 505 505 505 505 505 505 505 505 505
sum(x')
ans =
 505 505 505 505 505 505 505 505 505 505
sum(diag(x))
ans =
 505
sum(diag(fliplr(x)))
ans =
 505
```

Ejercicio de práctica 5.1

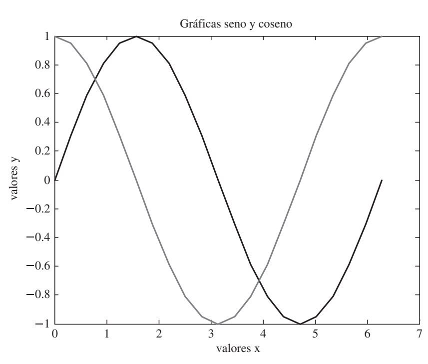
1.
```
clear,clc
x=0:0.1*pi:2*pi;
y=sin(x);
plot(x,y)
```



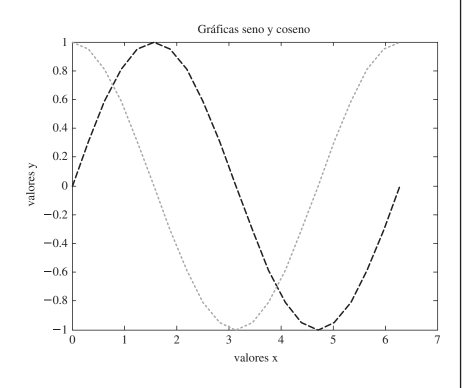
2.
```
title('Curva sinusoidal')
xlabel('valores x')
ylabel('sen(x)')
```



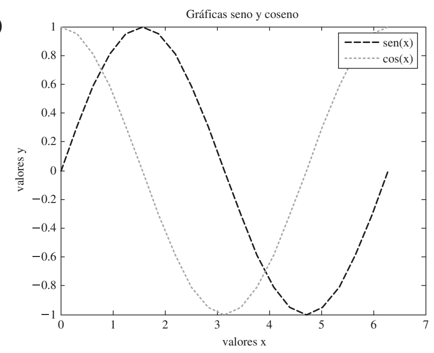
3.
```
figure(2)
y1=sin(x);
y2=cos(x);
plot(x,y1,x,y2)
title('Gráficas
  seno y coseno')
xlabel('valores x')
ylabel('valores y')
```



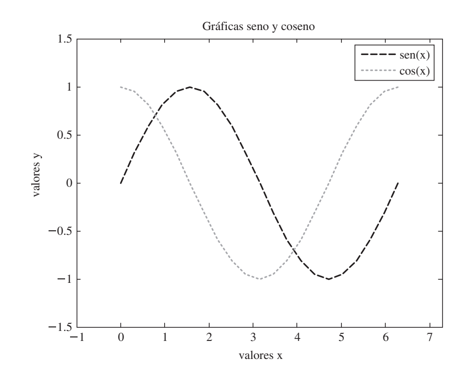
4.
```
figure(3)
plot(x,y1,'-- r',
  x,y2,': g')
title('Gráficas
  seno y coseno')
xlabel('valores x')
ylabel('valores y')
```



5. 
`legend('sen(x)','cos(x)')`

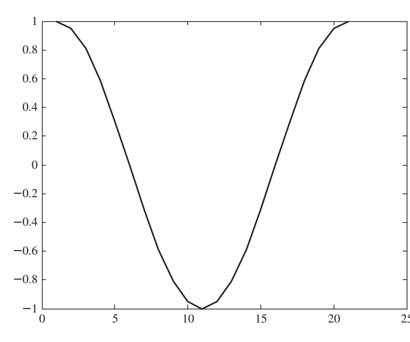


6. `axis([-1,2*pi+1,`
   `-1.5,1.5])`



7.
```
figure(4)
a=cos(x);
plot(a)
```

Se crea una gráfica de línea, con **a** graficada contra el número índice de vector.



Ejercicio de práctica 5.2

1. ```
   subplot(2,1,1)
   ```

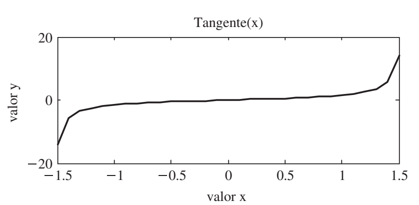
2. ```
   x=-1.5:0.1:1.5;
   y=tan(x);
   plot(x,y)
   ```

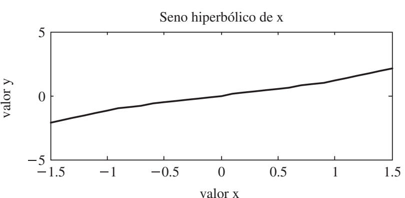
3. ```
   title('Tangente(x)')
   xlabel('valor x')
   ylabel('valor y')
   ```

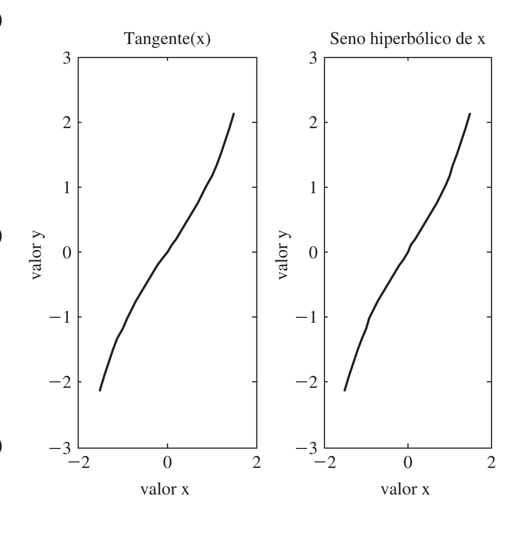
4. ```
   subplot(2,1,2)
   y=sinh(x);
   plot(x,y)
   ```

5. ```
   title('Seno
     hiperbólico de x')
   xlabel('valor x')
   ylabel('valor y')
   ```

6. ```
   figure(2)
   subplot(1,2,1)
   plot(x,y)
   title('Tangente(x)')
   xlabel('valor x')
   ylabel('valor y')
   subplot(1,2,2)
   y=sinh(x);
   plot(x,y)
   title('Seno
     hiperbólico de x')
   xlabel('valor x')
   ylabel('valor y')
   ```



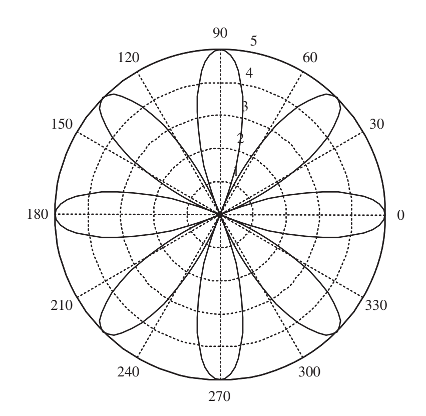




Ejercicio de práctica 5.3

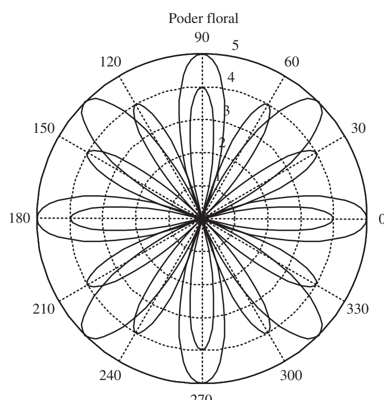
1.

```
theta = 0:0.01*pi:2*pi;
r = 5*cos(4*theta);
polar(theta,r)
```



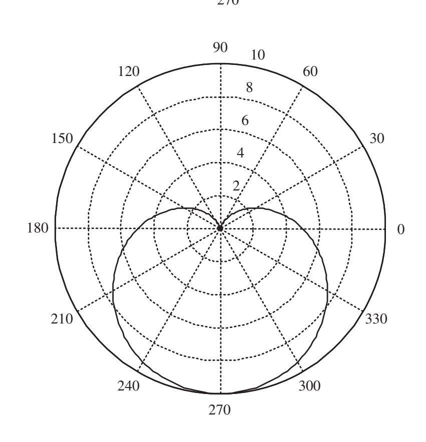
2.

```
hold on
r=4*cos(6*theta);
polar(theta,r)
title('Poder floral')
```



3.

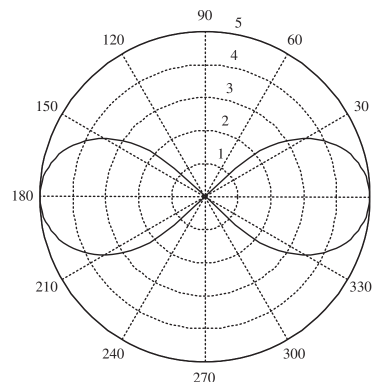
```
figure(2)
r=5-5*sin(theta);
polar(theta,r)
```



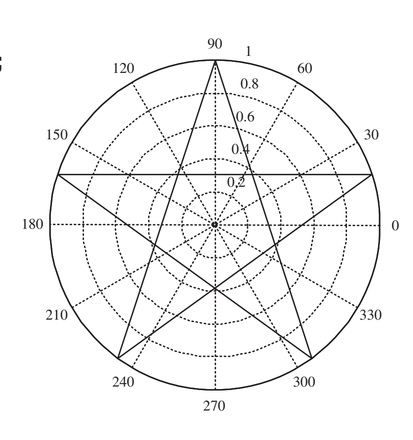
(Continúa)

Ejercicio de práctica 5.3 (*Continuación*)

4.
```
figure(3)
r = sqrt(5^2*cos(2*theta));
polar(theta3,r)
```



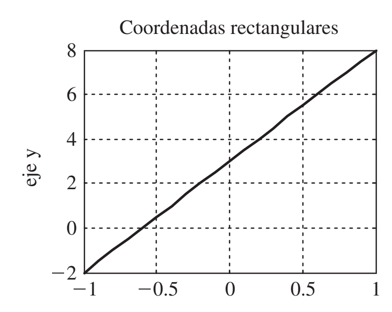
5.
```
figure(4)
theta = pi/2:4/5*pi:4.8*pi;
r=ones(1,6);
polar(theta,r)
```

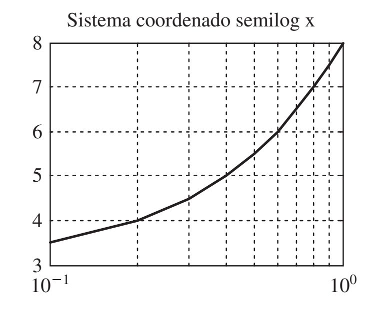


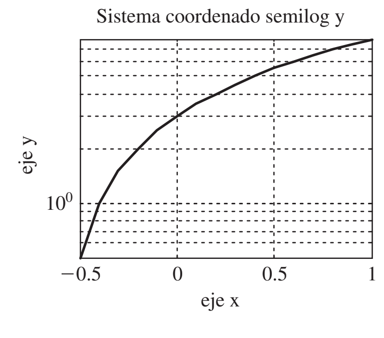
Ejercicio de práctica 5.4

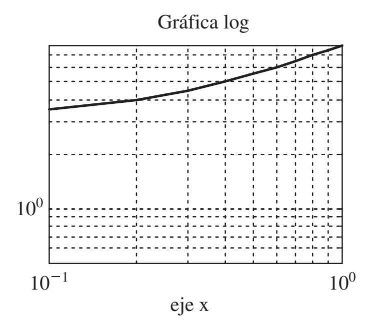
1.
```
figure(1)
x=-1:0.1:1;
y=5*x+3;
subplot(2,2,1)
plot(x,y)
title('Coordenadas rectangulares')
ylabel('eje y')
grid on
subplot(2,2,2)
semilogx(x,y)
title('Sistema coordenado semilog x')
grid on
subplot(2,2,3)
semilogy(x,y)
title('Sistema coordenado semilog y')
ylabel('eje y')
xlabel('eje x')
```

```
grid on
subplot(2,2,4)
loglog(x,y)
title('Gráfica log')
xlabel('eje x')
grid on
```





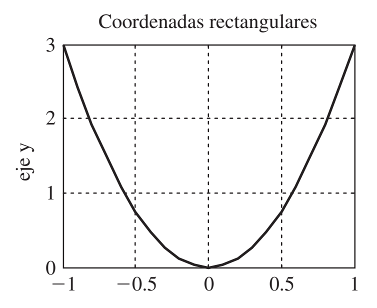


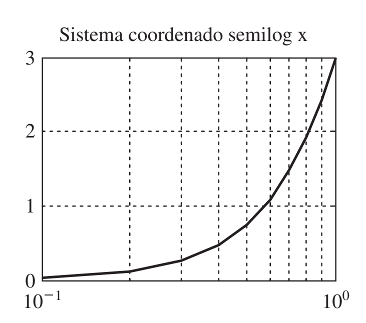


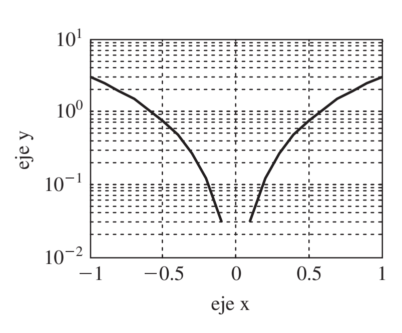
2.
```
figure(2)
x=-1:0.1:1;
y=3*x.^2;
subplot(2,2,1)
plot(x,y)
title('Coordenadas rectangulares')
ylabel('eje y')
grid on
subplot(2,2,2)
semilogx(x,y)
title('Sistema coordenado semilog x')
grid on
subplot(2,2,3)
semilogy(x,y)
title('Sistema coordenado semilog y')
ylabel('eje y')
xlabel('eje x')
grid on
subplot(2,2,4)
loglog(x,y)
title('Gráfica log')
xlabel('eje x')
grid on
```

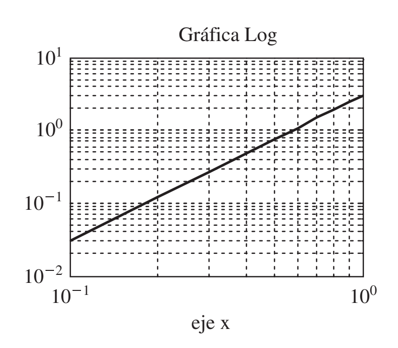
(Continúa)

Ejercicio de práctica 5.4 (*Continuación*)

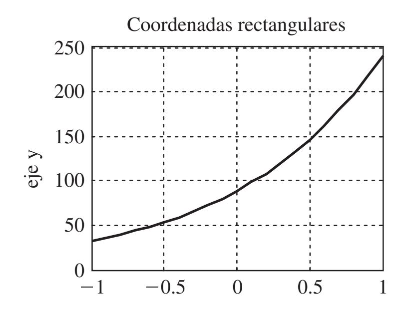


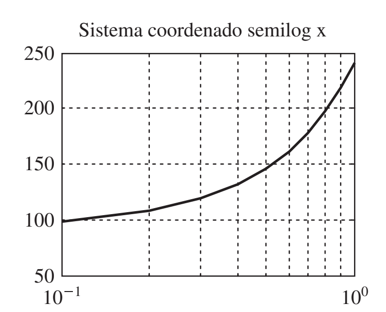


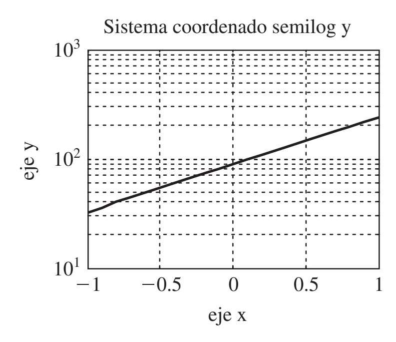


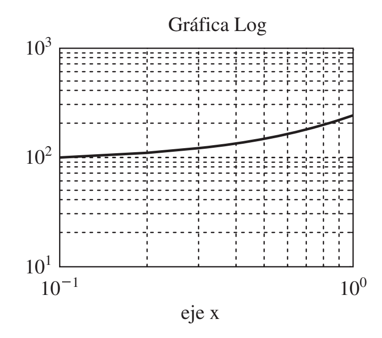


3.
```
figure(3)
x=-1:0.1:1;
y=12*exp(x+2);
subplot(2,2,1)
plot(x,y)
title('Coordenadas rectangulares')
ylabel('eje y')
grid on
subplot(2,2,2)
semilogx(x,y)
title('Sistema coordenado semilog x')
grid on
subplot(2,2,3)
semilogy(x,y)
title('Sistema coordenado semilog y')
ylabel('eje y')
xlabel('eje x')
grid on
subplot(2,2,4)
loglog(x,y)
title('Gráfica Log')
xlabel('eje x')
grid on
```









4.
```
figure(4)
x=-1:0.01:1;
y=1./x;
subplot(2,2,1)
plot(x,y)
title('Coordenadas rectangulares')
ylabel('eje y')
grid on
subplot(2,2,2)
semilogx(x,y)
title('Sistema coordenado semilog x')
grid on
subplot(2,2,3)
semilogy(x,y)
title('Sistema coordenado semilog y')
ylabel('eje y')
xlabel('eje x')
grid on
subplot(2,2,4)
loglog(x,y)
title('Gráfica Log')
xlabel('eje x')
grid on
```

(Continúa)

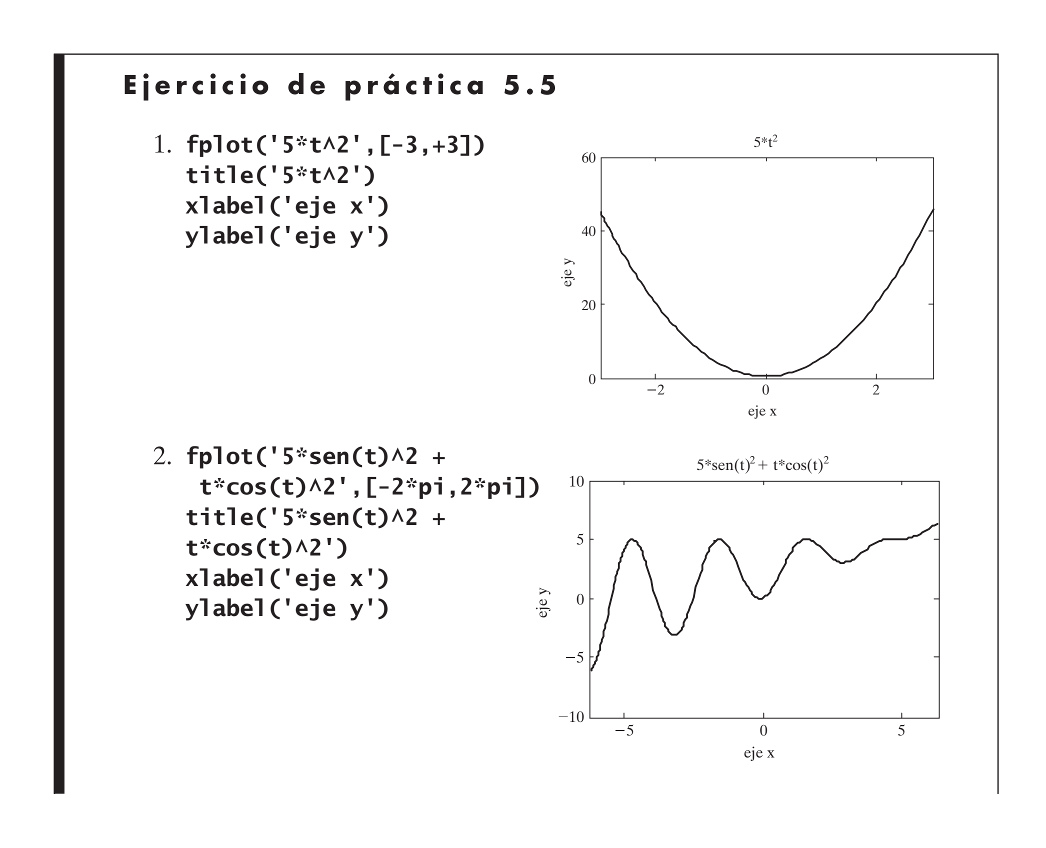
Ejercicio de práctica 5.4 (*Continuación*)

Coordenadas rectangulares

Sistema coordenado semilog x

Sistema coordenado semilog y

Gráfica Log



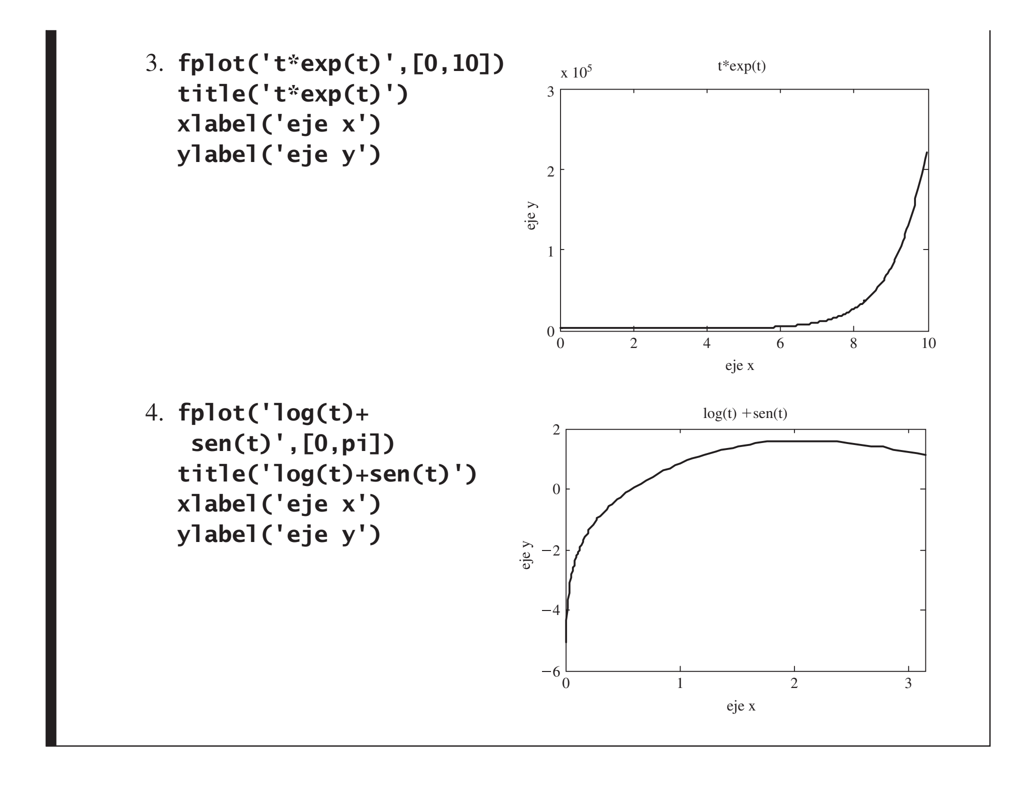
Ejercicio de práctica 5.5

1. ```
   fplot('5*t^2',[-3,+3])
   title('5*t^2')
   xlabel('eje x')
   ylabel('eje y')
   ```

2. ```
   fplot('5*sen(t)^2 +
    t*cos(t)^2',[-2*pi,2*pi])
   title('5*sen(t)^2 +
   t*cos(t)^2')
   xlabel('eje x')
   ylabel('eje y')
   ```

3.
```
fplot('t*exp(t)',[0,10])
title('t*exp(t)')
xlabel('eje x')
ylabel('eje y')
```

4.
```
fplot('log(t)+
 sen(t)',[0,pi])
title('log(t)+sen(t)')
xlabel('eje x')
ylabel('eje y')
```



Ejercicio de práctica 6.1

Almacene estas funciones como archivos-m separados. El nombre de la función debe ser el mismo que el nombre del archivo-m.

1.
```
function output = quad(x)
output = x.^2;
```
2.
```
function output=one_over(x)
output = exp(1./x);
```
3.
```
function output = sin_x_squared(x)
output = sin(x.^2);
```
4.
```
function result = in_to_ft(x)
result = x./12;
```
5.
```
function result=cal_to_joules(x)
result = 4.2.*x;
```
6.
```
function output = Watts_to_Btu_per_hour(x)
output = x.*3.412;
```
7.
```
function output = meters_to_miles(x)
output = x./1000.*.6214;
```
8.
```
function output = mph_to_fps(x)
output = x.*5280/3600;
```

Ejercicio de práctica 6.2

Almacene estas funciones como archivos-m separados. El nombre de la función debe ser el mismo que el nombre del archivo-m.

1.
```
function output = z1(x,y)
% suma de x y y
% las dimensiones de la matriz deben concordar
output = x+y;
```
2.
```
function output = z2(a,b,c)
% encuentra a.*b.^c
% las dimensiones de la matriz deben concordar
output = a.*b.^c;
```
3.
```
function output = z3(w,x,y)
% encuentra w.*exp(x./y)
% las dimensiones de la matriz deben concordar
output = w.*exp(x./y);
```
4.
```
function output = z4(p,t)
% encuentra p./sin(t)
% las dimensiones de la matriz deben concordar
output = p./sin(t);
```
5.
```
function [a,b]=f5(x)
a = cos(x);
b = sin(x);
```
6.
```
function [a,b] = f6(x)
a = 5.*x.^2 + 2;
b = sqrt(5.*x.^2 + 2);
```
7.
```
function [a,b] = f7(x)
a = exp(x);
b = log(x);
```
8.
```
function [a,b] = f8(x,y)
a = x+y;
b = x-y;
```
9.
```
function [a,b] = f9(x,y)
a = y.*exp(x);
b = x.*exp(y);
```

Ejercicio de práctica 7.1

1.
```
b = input('Ingrese la longitud de la base del
          triángulo: ');
h = input('Ingrese la altura del triángulo: ');
Area = 1/2*b*h
```
Cuando este archivo corre, genera la siguiente interacción en la ventana de comandos:
```
Enter the length of the base of the triangle: 5
Enter the height of the triangle: 4
Area =
  10
```
2.
```
r = input('Ingrese el radio del cilindro: ');
h = input('Ingrese la altura del cilindro: ');
Volume = pi*r.^2*h
```
Cuando este archivo corre, genera la siguiente interacción en la ventana de comandos:
```
Enter the radius of the cylinder: 2
Enter the height of the cylinder: 3
Volume =
 37.6991
```
3.
```
n = input('Ingrese un valor de n: ')
vector = 0:n
```
Cuando este archivo corre, genera la siguiente interacción en la ventana de comandos:
```
Enter a value of n: 3
n =
   3
vector =
   0 1 2 3
```
4.
```
a = input('Ingrese el valor inicial: ');
b = input('Ingrese el valor final: ');
c = input('Ingrese el espaciamiento del vector: ');
vector = a:c:b
```
Cuando este archivo corre, genera la siguiente interacción en la ventana de comandos:
```
Enter the starting value: 0
Enter the ending value: 6
Enter the vector spacing: 2
vector =
   0 2 4 6
```

Ejercicio de práctica 7.2

1. `disp('Tabla de conversión de pulgadas a pies')`
2. `disp('  Pulgadas   Pies')`
3.
```
inches = 0:10:120;
feet = inches./12;
table = [inches; feet];
fprintf(' %8.0f %8.2f \n',table)
```
   El despliegue resultante en la ventana de comandos es

   Tabla de conversión de pulgadas a pies

Pulgadas	Pies
0	0.00
10	0.83
20	1.67
30	2.50
40	3.33
50	4.17
60	5.00
70	5.83
80	6.67
90	7.50
100	8.33
110	9.17
120	10.00

Ejercicio de práctica 8.1

Use estos arreglos en los ejercicios.

```
x=[1 10 42 6
   5 8 78 23
   56 45 9 13
   23 22 8 9];
y=[1 2 3; 4 10 12; 7 21 27];
z=[10 22 5 13];
```

1.
```
elements_x = find(x>10)
elements_y = find(y>10)
elements_z = find(z>10)
```
2.
```
[rows_x, cols_x]=find(x>10)
[rows_y, cols_y]=find(y>10)
[rows_z, cols_z]=find(z>10)
```
3.
```
x(elements_x)
y(elements_y)
z(elements_z)
```

4.
```
elements_x = find(x>10 & x< 40)
elements_y = find(y>10 & y< 40)
elements_z = find(z>10 & z< 40)
```

5.
```
[rows_x, cols_x]=find(x>10 & x<40)
[rows_y, cols_y]=find(y>10 & y<40)
[rows_z, cols_z]=find(z>10 & z<40)
```

6.
```
x(elements_x)
y(elements_y)
z(elements_z)
```

7.
```
elements_x = find((x>0 & x<10) | (x>70 & x<80))
elements_y = find((y>0 & y<10) | (y>70 & y<80))
elements_z = find((z>0 & z<10) | (z>70 & z<80))
```

8.
```
length_x = length(find((x>0 & x<10) | (x>70 & x<80)))
length_y = length(find((y>0 & y<10) | (y>70 & y<80)))
length_z = length(find((z>0 & z<10) | (z>70 & z<80)))
```

Ejercicio de práctica 8.2

1.
```
function output = drink(x)
if x>=21
   output = 'Puede beber';
else
   output = 'Espere hasta ser mayor';
end
```

Pruebe su función con lo siguiente:

```
drink(22)
drink(18)
```

2.
```
function output = tall(x)
if x>=48
   output='Puede subir';
else
   output = 'Es muy bajo';
end
```

Pruebe su función con lo siguiente:

```
tall(50)
tall(46)
```

3.
```
function output = spec(x)
if x>=5.3 & x<=5.5
   output = 'dentro de las espe';
else
   output = ' fuera de las espe';
end
```

Pruebe su función con lo siguiente:

```
spec(5.6)
spec(5.45)
spec(5.2)
```

(Continúa)

Ejercicio de práctica 8.2 (*Continuación*)

4.
```
function output = metric_spec(x)
if x>=5.3/2.54 & x<=5.5/2.54
   output = 'dentro de las espe';
else
   output = ' fuera de las espe';
end
```
Pruebe su función con lo siguiente:
```
metric_spec(2)
metric_spec(2.2)
metric_spec(2.4)
```

5.
```
function output = flight(x)
if x>=0 & x<=100
   output='primera etapa';
elseif x<=170
   output = 'segunda etapa';
elseif x<260
   output = 'tercera etapa';
else
   output = 'caída libre';
end
```
Pruebe su función con lo siguiente:
```
flight(50)
flight(110)
flight(200)
flight(300)
```

Ejercicio de práctica 8.3

1.
```
year = input('Ingrese el nombre de su año
       en la escuela: ','s');
switch year
  case 'primero'
    day='Lunes';
  case 'segundo'
    day = 'Martes';
  case 'tercero'
    day = 'Miércoles';
  case 'cuarto'
    day = 'Jueves';
  otherwise
    day = 'No sé el año';
end
disp(['Sus finales son el ',day])
```

2.
```
disp('¿En qué año está en la escuela?')
disp('Use el menú para hacer su selección ')
   choice = menu('Año en la escuela','primero','segundo',
                 'tercero', 'cuarto');
```

```
switch choice
  case 1
    day = 'Lunes';
  case 2
    day = 'Martes';
  case 3
    day = 'Miércoles';
  case 4
    day = 'Jueves';
end
disp(['Sus finales son el',day])
```

3.
```
num = input('¿Cuántas barras de dulce quiere?');
switch num
  case 1
    bill = 0.75;
  case 2
    bill = 1.25;
  case 3
    bill = 1.65;
  otherwise
    bill = 1.65 + (num-3)*0.30;
end
fprintf('Su importe es %5.2f \n',bill)
```

Ejercicio de práctica 8.4

1.
```
inches = 0:3:24;
for k=1:length(inches)
  feet(k) = inches(k)/12;
end
table=[inches',pies']
```

2.
```
x = [ 45,23,17,34,85,33];
count=0;
for k=1:length(x)
  if x(k)>30
     count = count+1;
  end
end
fprintf('Existen %4.0f valores mayores que 30 \n',count)
```

3.
```
num = length(find(x>30));
fprintf('Existen %4.0f valores mayores que 30 \n',num)
```

4.
```
total = 0;
for k=1:length(x)
    total = total + x(k);
end
disp('El total es:')
disp(total)
sum(x)
```

Ejercicio de práctica 8.5

1.
```
inches = 0:3:24;
k=1;
while k<=length(inches)
   feet(k) = inches(k)/12;
   k=k+1;
end
disp(' Pulgadas Pies');
fprintf(' %8.0f %8.2f \n',[inches;feet])
```
2.
```
x = [ 45,23,17,34,85,33];
k=1;
count = 0;
while k<=length(x)
 if x(k)>=30;
   count = count +1;
 end
   k=k+1;
end
fprintf('Existen %4.0f valores mayores que
         30 \n',count)
```
3.
```
count = length(find(x>30))
```
4.
```
k=1;
total = 0;
while k<=length(x)
   total = total + x(k);
   k=k+1;
end
disp(total)
sum(x)
```

Ejercicio de práctica 9.1

1.
```
A = [ 1 2 3 4]
B = [ 12 20 15 7]
dot(A,B)
```
2.
```
sum(A.*B)
```
3.
```
price=[0.99, 1.49, 2.50, 0.99, 1.29];
num = [4, 3, 1, 2, 2];
total=dot(price,num)
```

Ejercicio de práctica 9.2

1.
```
A=[2 5; 2 9; 6 5];
B=[2 5; 2 9; 6 5];
% Éstos no se pueden multiplicar porque el número de
% columnas en A no es igual al número de filas en B
```
2.
```
A=[2 5; 2 9; 6 5];
B=[1 3 12; 5 2 9];
% Dado que A es una matriz 3 × 2 y B es una matriz 2 × 3
% se pueden multiplicar
A*B
%Sin embargo, A*B no es igual a B*A
B*A
```
3.
```
A=[5 1 9; 7 2 2];
B = [8 5; 4 2; 8 9];
% Dado que A es una matriz 2 × 3 y B es una matriz 3 × 2,
% se pueden multiplicar
A*B
%Sin embargo, A*B no es igual a B*A
B*A
```
4.
```
A=[1 9 8; 8 4 7; 2 5 3];
B=[7;1;5]
% Dado que A es una matriz 3 × 3 y B es una matriz 3 × 1,
% se pueden multiplicar
A*B
%  Sin embargo, B*A no funcionará
```

Ejercicio de práctica 9.3

1. a.
```
a=magic(3)
inv(magic(3))
magic(3)^-1
```
   b.
```
b=magic(4)
inv(b)
b^-1
```
   c.
```
c=magic(5)
inv(magic(5))
magic(5)^-1
```
2.
```
det(a)
det(b)
det(c)
```
3.
```
A=[1 2 3;2 4 6;3 6 9]
det(A)
inv(A)
%Note que las tres líneas sólo son múltiplos uno del
%otro y por tanto no representan ecuaciones
  independientes
```

Ejercicio de práctica 10.1

1.
```
A = [1,4,6; 3, 15, 24; 2, 3,4];
B=single(A)
C=int8(A)
D=uint8(A)
```
2.
```
E = A+B
% El resultado es un arreglo de precisión sencilla
```
3.
```
x=int8(1)
y=int8(3)
result1=x./y
% Este cálculo regresa el entero 0
x=int8(2)
result2=x./y

% Este cálculo regresa el entero 1; parece
% que MATLAB redondea la respuesta
```
4.
```
intmax('int8')
intmax('int16')
intmax('int32')
intmax('int64')
intmax('uint8')
intmax('uint16')
intmax('uint32')
intmax('uint64')
```
5.
```
intmin('int8')
intmin('int16')
intmin('int32')
intmin('int64')
intmin('uint8')
intmin('uint16')
intmin('uint32')
intmin('uint64')
```

Ejercicio de práctica 10.2

1.
```
name ='Holly'
```
2.
```
G=double('g')
fprintf('El equivalente decimal de la letra g es %5.0f
        \n',G)
```
3.
```
m='MATLAB'
M=char(double(m)-32)
```

Ejercicio de práctica 10.3

1. ```
   a=magic(3)
   b=zeros(3)
   c=ones(3)
   x(:,:,1)=a
   x(:,:,2)=b
   x(:,:,3)=c
   ```
2. `x(3,2,1)`
3. `x(2,3,:)`
4. `x(:,3,:)`

Ejercicio de práctica 10.4

1. ```
   names=char('Mercurio','Venus','Tierra','Marte','Jupiter',
               'Saturno','Urano','Neptuno','Plutón')
   ```
2. ```
   R='rocky';
   G='gas giants';
   type=char(R,R,R,R,G,G,G,G,R)
   ```
3. `space =[' ';' ';' ';' ';' ';' ';' ';' ';' '];`
4. `table =[names,space,type]`
5. ```
   %Estos datos se encontraron en
   %  http://sciencepark.etacude.com/astronomy/pluto.php
   %Datos similares se encuentran en muchos websites
   mercury=3.303e23; % kg
   venus = 4.869e24; % kg
   earth = 5.976e24; % kg
   mars = 6.421e23; % kg
   jupiter=1.9e27; % kg
   saturn = 5.69e26; % kg
   uranus = 8.686e25; % kg
   neptune = 1.024e26; % kg
   pluto = 1.27e22 % kg
   mass = [mercury,venus,earth,mars,jupiter,
           saturn,uranus,neptune,pluto]';
   newtable=[table,space,num2str(mass)]
   ```

Ejercicio de práctica 11.1

1.
```
syms x a b c
%or
d=sym('d') %etc
d =
d
```
2.
```
ex1 = x^2-1
ex1 =
x^2-1
ex2 = (x+1)^2
ex2 =
(x+1)^2
ex3 = a*x^2-1
ex3 =
a*x^2-1
ex4 = a*x^2 + b*x + c
ex4 =
a*x^2+b*x+c
ex5 = a*x^3 + b*x^2 + c*x + d
ex5 =
a*x^3+b*x^2+c*x+d
ex6 = sin(x)
ex6 =
sin(x)
```
3.
```
EX1 = sym('X^2 - 1 ')
EX1 =
X^2 - 1
EX2 = sym(' (X +1)^2 ')
EX2 =
(X +1)^2
EX3 = sym('A*X ^2 - 1 ')
EX3 =
A*X ^2 - 1
EX4 = sym('A*X ^2 + B*X + C ')
EX4 =
A*X ^2 + B*X + C
EX5 = sym('A*X ^3 + B*X ^2 + C*X + D ')
EX5 =
A*X ^3 + B*X ^2 + C*X + D
EX6 = sym(' sin(X) ')
EX6 =
sin(X)
```
4.
```
eq1= sym(' x^2=1 ')
eq1 =
x^2=1
eq2= sym(' (x+1)^2=0 ')
eq2 =
(x+1)^2=0
```

```
eq3= sym(' a*x^2=1 ')
eq3 =
a*x^2=1
eq4 = sym('a*x^2 + b*x + c=0 ')
eq4 =
a*x^2 + b*x + c=0
eq5 = sym('a*x^3 + b*x^2 + c*x + d=0 ')
eq5 =
a*x^3 + b*x^2 + c*x + d=0
eq6 = sym('sin(x)=0 ')
eq6 =
sin(x)=0
```

5.
```
EQ1 = sym('X^2 = 1 ')
EQ1 =
X^2 = 1
EQ2 = sym(' (X +1)^2=0 ')
EQ2 =
(X +1)^2=0
EQ3 = sym('A*X ^2 =1 ')
EQ3 =
A*X ^2 =1
EQ4 = sym('A*X ^2 + B*X + C = 0 ')
EQ4 =
A*X ^2 + B*X + C = 0
EQ5 = sym('A*X ^3 + B*X ^2 + C*X + D = 0 ')
EQ5 =
A*X ^3 + B*X ^2 + C*X + D = 0
EQ6 = sym(' sin(X) = 0 ')
EQ6 =
sin(X) = 0
```

Ejercicio de práctica 11.2

1.
```
y1=ex1*ex2
y1 =
(x^2-1)*(x+1)^2
```

2.
```
y2=ex1/ex2
y2 =
(x^2-1)/(x+1)^2
```

3.
```
[num1,den1]=numden(y1)
num1 =
(x^2-1)*(x+1)^2
den1 =
1
```

(Continúa)

Ejercicio de práctica 11.2 (*Continuación*)

```
[num2,den2]=numden(y2)
num2 =
x^2-1
den2 =
(x+1)^2
```

4. ```
   Y1=EX1*EX2
   Y1 =
   (X^2-1)*(X+1)^2
   ```
5. ```
   Y2=EX1/EX2
   Y2 =
   (X^2-1)/(X+1)^2
   ```
6. ```
   [NUM1,DEN1]=numden(Y1)
   NUM1 =
   (X^2-1)*(X+1)^2
   DEN1 =
   1
   [NUM2,DEN2]=numden(Y2)
   NUM2 =
   X^2-1
   DEN2 =
   (X+1)^2
   ```
7. ```
   %numden(EQ4)
   %La función numden no se aplica a ecuaciones,
   %sólo a expresiones
   ```
8. **a.**
   ```
   factor(y1)
   ans =
   (x-1)*(x+1)^3
   expand(y1)
   ans =
   x^4+2*x^3-2*x-1
   collect(y1)
   ans =
   x^4+2*x^3-2*x-1
   ```
   b.
   ```
   factor(y2)
   ans =
   (x-1)/(x+1)
   expand(y2)
   ans =
   1/(x+1)^2*x^2-1/(x+1)^2
   collect(y2)
   ans =
   (x^2-1)/(x+1)^2
   ```
   c.
   ```
   factor(Y1)
   ans =
   (X-1)*(X+1)^3
   expand(Y1)
   ```

```
ans =
X^4+2*X^3-2*X-1
collect(Y1)
ans =
X^4+2*X^3-2*X-1
```

d. `factor(Y2)`

```
ans =
(X-1)/(X+1)
expand(Y2)
ans =
1/(X+1)^2*X^2-1/(X+1)^2
collect(Y2)
ans =
(X^2-1)/(X+1)^2
```

9. `factor(ex1)`

```
ans =
(x-1)*(x+1)
expand(ex1)
ans =
x^2-1
collect(ex1)
ans =
x^2-1
factor(eq1)
ans =
x^2 = 1
expand(eq1)
ans =
x^2 = 1
collect(eq1)
ans =
x^2 = 1
%
factor(ex2)
ans =
(x+1)^2
expand(ex2)
ans =
x^2+2*x+1
collect(ex2)
ans =
x^2+2*x+1
factor(eq2)
ans =
(x+1)^2 = 0
expand(eq2)
ans =
x^2+2*x+1 = 0
collect(eq2)
ans =
x^2+2*x+1 = 0
```

Ejercicio de práctica 11.3

1.
```
solve(ex1)
ans =
 1
-1
solve(EX1)
ans =
 1
-1
solve(eq1)
ans =
 1
-1
solve(EQ1)
ans =
 1
-1
```

2.
```
solve(ex2)
ans =
-1
-1
solve(EX2)
ans =
-1
-1
solve(eq2)
ans =
 -1
 -1
solve(EQ2)
ans =
-1
-1
```

3. a.
```
A=solve(ex3,x,a)
Warning: 1 equations in 2 variables.
A =
   a: [1x1 sym]
   x: [1x1 sym]
A.a
ans =
1/x^2
A.x
ans =
x
%or
[my_a,my_x]=solve(ex3,x,a)
Warning: 1 equations in 2 variables.
my_a =
1/x^2
```

```
my_x =
x
```

b.
```
A=solve(eq3,x,a)
Warning: 1 equations in 2 variables.
A =
  a: [1x1 sym]
   x: [1x1 sym]
A.a
ans =
1/x^2
A.x
ans =
x
```

4. **a.**
```
A=solve(EX3,'X','A')
Warning: 1 equations in 2 variables.
A =
  A: [1x1 sym]
 X: [1x1 sym]
A.A
ans =
1/X^2
A.X
ans =
X
%o
[My_A,My_X]=solve(EX3,'X','A')
Warning: 1 equations in 2 variables.
My_A =
1/X^2
My_X =
X
```

b.
```
A=solve(EQ3,'X','A')
Warning: 1 equations in 2 variables.
A =
  A: [1x1 sym]
 X: [1x1 sym]
A.A
ans =
1/X^2
A.X
ans =
X
```

5. **a.**
```
A=solve(ex4,x,a)
Warning: 1 equations in 2 variables.
A =
  a: [1x1 sym]
  x: [1x1 sym]
A.a
```

(Continúa)

Ejercicio de práctica 11.3 (*Continuación*)

```
ans =
-(b*x+c)/x^2
A.x
ans =
x
%o
[my_a,my_x]=solve(ex4,x,a)
Warning: 1 equations in 2 variables.
my_a =
-(b*x+c)/x^2
my_x =
x
%b
```

b.
```
A=solve(eq4,x,a)
Warning: 1 equations in 2 variables.
A =
  a: [1x1 sym]
  x: [1x1 sym]
A.a
ans =
-(b*x+c)/x^2
A.x
ans =
x
```

6. a.
```
A=solve(EX4,'X','A')
Warning: 1 equations in 2 variables.
A =
  A: [1x1 sym]
  X: [1x1 sym]
A.A
ans =
-(B*X+C)/X^2
A.X
ans =
X
%o
[My_A,My_X]=solve(EX4,'X','A')
Warning: 1 equations in 2 variables.
My_A =
-(B*X+C)/X^2
My_X =
X
```

b.
```
A=solve(EQ4,'X','A')
Warning: 1 equations in 2 variables.
A =
 A: [1x1 sym]
 X: [1x1 sym]
```

```
A.A
ans =
-(B*X+C)/X^2
A.X
ans =
X
```

7. **A=solve(ex5,x)**

```
A =
1/6/a*(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*(4*c^3*a-c^2*b^2-
18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3)-2/3*(3*c*a-
b^2)/a/(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*(4*c^3*a-c^2*b^2-
18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3)-1/3*b/a
-1/12/a*(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*(4*c^3*a-c^2*b^2-
18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3)+1/3*(3*c*a-
b^2)/a/(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*(4*c^3*a-c^2*b^2-
18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3)-
1/3*b/a+1/2*i*3^(1/2)*(1/6/a*(36*c*b*a-108*d*a^2-
8*b^3+12*3^(1/2)*(4*c^3*a-c^2*b^2-
18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3)+2/3*(3*c*a-
b^2)/a/(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*(4*c^3*a-c^2*b^2-
18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3))
-1/12/a*(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*(4*c^3*a-c^2*b^2-
18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3)+1/3*(3*c*a-
b^2)/a/(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*(4*c^3*a-c^2*b^2-
18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3)-1/3*b/a-
1/2*i*3^(1/2)*(1/6/a*(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*
(4*c^3*a-c^2*b^2-18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3)
+2/3*(3*c*a-b^2)/a/(36*c*b*a-108*d*a^2-8*b^3+12*3^(1/2)*
(4*c^3*a-c^2*b^2-18*c*b*a*d+27*d^2*a^2+4*d*b^3)^(1/2)*a)^(1/3))
% Claramente esto es demasiado complejo para memorizar
```

8. **solve(ex6)**

```
ans =
0
solve(EX6)
ans =
0
solve(eq6)
ans =
0
solve(EQ6)
ans =
0
```

Ejercicio de práctica 11.4

1.
```
A=sym('x^2 +5*y -3*z^3=15')
A =
x^2 +5*y -3*z^3=15
B=sym('4*x + y^2 -z = 10')
B =
4*x + y^2 -z = 10
C=sym('x + y + z =15')
C =
x + y + z =15
[X,Y,Z]=solve(A,B,C)
X =
11.560291920108418818149999909102-
11.183481663794727000635376340336*i
10.217253727895446083582447731954-
4.7227311648148859415297821017854*i
16.889121018662801764934219025612-
4.2177563835168647657970523110679*i
16.889121018662801764934219025612+4.2177563835168647657970523110679*i
10.217253727895446083582447731954+4.7227311648148859415297821017854*i
11.560291920108418818149999909102+11.183481663794727000635376340336*i
Y =
3.5094002752389020636845577121798+6.9732883324603664143501389722123*i
1.6407253627272400391783345506916+5.5153398551767327926224186488990*i
.84987436203385789713710773712855+7.8113869374516535777173651228485*i
.84987436203385789713710773712855-
7.8113869374516535777173651228485*i
 1.6407253627272400391783345506916-
5.5153398551767327926224186488990*i
 3.5094002752389020636845577121798-
6.9732883324603664143501389722123*i
Z =
 -.69692195347320881834557621281 4e-
1+4.2101933313343605862852373681236*i
 3.1420209093773138772392177173549-
.79260869036184685109263654711 36*i
 -2.7389953806966596620713267627408-
3.5936305539347888119203128117806*i
2.-7389953806966596620713267627408+3.5936305539347888119203128117806*i
```

```
3.14202090937731387723921771735 49+.79260869036184685109263654711 36
*i
     -.69692195347320881834557621 2814e-1-
    4.21019333133436058628523736 81236*i
    double(X)
    ans =
     11.5603 -11.1835i
     10.2173 - 4.7227i
     16.8891 - 4.2178i
     16.8891 + 4.2178i
     10.2173 + 4.7227i
     11.5603 +11.1835i
    double(Y)
    ans =
      3.5094 + 6.9733i
      1.6407 + 5.5153i
      0.8499 + 7.8114i
      0.8499 - 7.8114i
      1.6407 - 5.5153i
      3.5094 - 6.9733i
    double(Z)
    ans =
     -0.0697 + 4.2102i
      3.1420 - 0.7926i
     -2.7390 - 3.5936i
     -2.7390 + 3.5936i
      3.1420 + 0.7926i
     -0.0697 - 4.2102i
```

Ejercicio de práctica 11.5

1.
```
eq1
eq1 =
x^2=1
subs(eq1,x,4)
ans =
16 = 1
ex1
ex1 =
x^2-1
subs(ex1,x,4)
ans =
  15
EQ1
```

(Continúa)

Ejercicio de práctica 11.5 (*Continuación*)

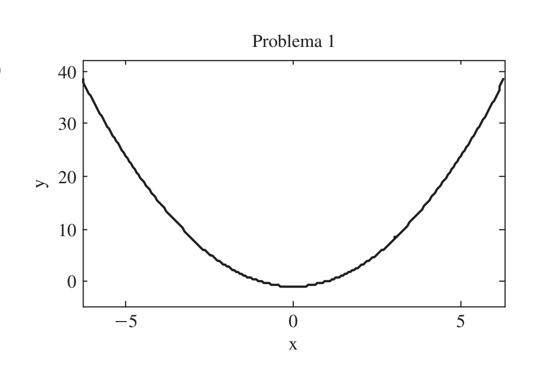
```
EQ1 =
X^2 = 1
subs(EQ1,'X',4)
ans =
16 = 1
EX1
EX1 =
X^2 - 1
subs(EX1,'X',4)
ans =
  15
% etc
```

2.
```
v=0:2:10;
subs(ex1,x,v)
ans =
 -1 3 15 35 63 99
subs(EX1,'X',v)
ans =
 -1 3 15 35 63 99
%subs(eq1,x,v)
%subs(EQ1,'X',v)
% No puede sustituir un vector en una ecuación
```

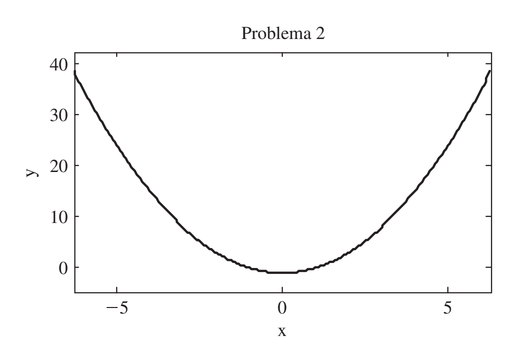
3.
```
new_ex1=subs(ex1,{a,b,c},{3,4,5})
new_ex1 =
x^2-1
subs(new_ex1,x,1:0.5:5)
ans =
 Columns 1 through 5
     0 1.2500 3.0000 5.2500 8.0000
 Columns 6 through 9
  11.2500 15.0000 19.2500 24.0000
new_EX1=subs(EX1,{'A','B','C'},{3,4,5})
new_EX1 =
X^2-1
subs(new_EX1,'X',1:0.5:5)
ans =
 Columns 1 through 5
     0 1.2500 3.0000 5.2500 8.0000
 Columns 6 through 9
  11.2500 15.0000 19.2500 24.0000
%
new_eq1=subs(eq1,{a,b,c},{3,4,5})
new_eq1 =
x^2 = 1
%subs(new_eq1,x,1:0.5:5) % no funcionará porque es una
%ecuación
new_EQ1=subs(EQ1,{'A','B','C'},{3,4,5})
new_EQ1 =
X^2=1
```

Ejercicio de práctica 11.6

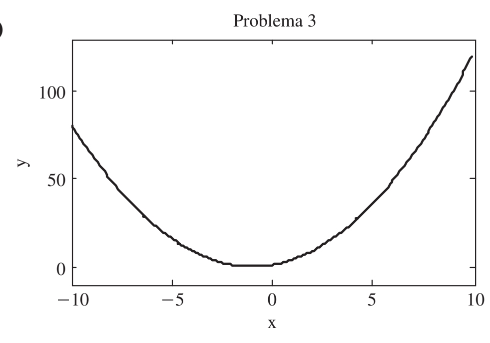
1.
```
ezplot(ex1)
title('Problema 1')
xlabel('x')
ylabel('y')
```



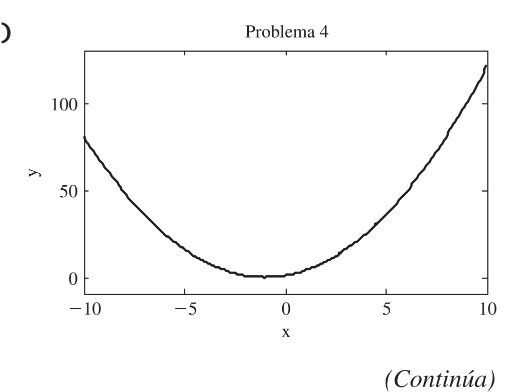
2.
```
ezplot(EX1)
title('Problema 2')
xlabel('x')
ylabel('y')
```



3.
```
ezplot(ex2,[-10,10])
title('Problema 3')
xlabel('x')
ylabel('y')
```



4.
```
ezplot(EX2,[-10,10])
title('Problema 4')
xlabel('x')
ylabel('y')
```

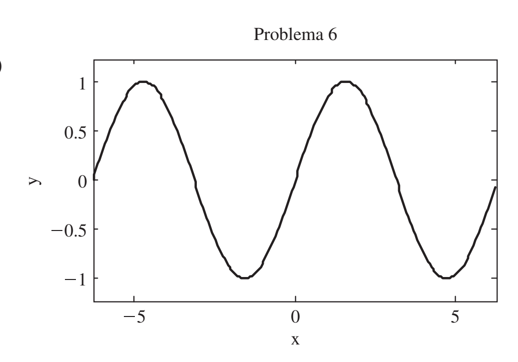


(Continúa)

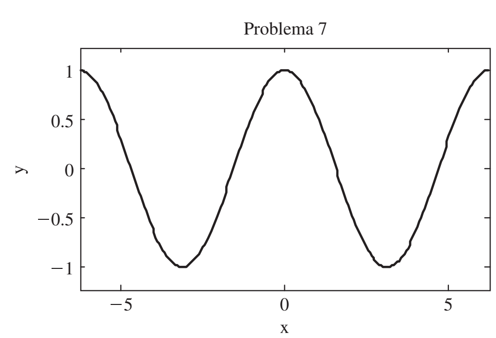
Ejercicio de práctica 11.6 ***(Continuación)***

5. Las ecuaciones con una sola variable tienen un solo valor válido de x; no hay pares x, y.

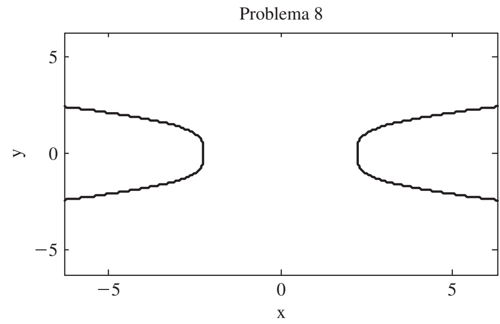
6.
```
ezplot(ex6)
title('Problema 6')
xlabel('x')
ylabel('y')
```



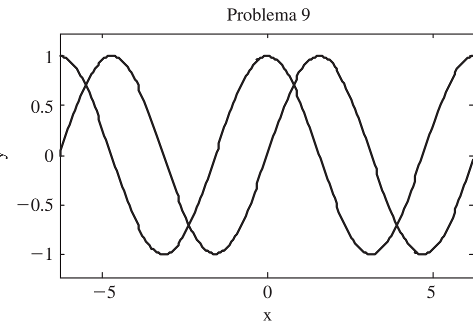
7.
```
ezplot('cos(x)')
title('Problema 7')
xlabel('x')
ylabel('y')
```



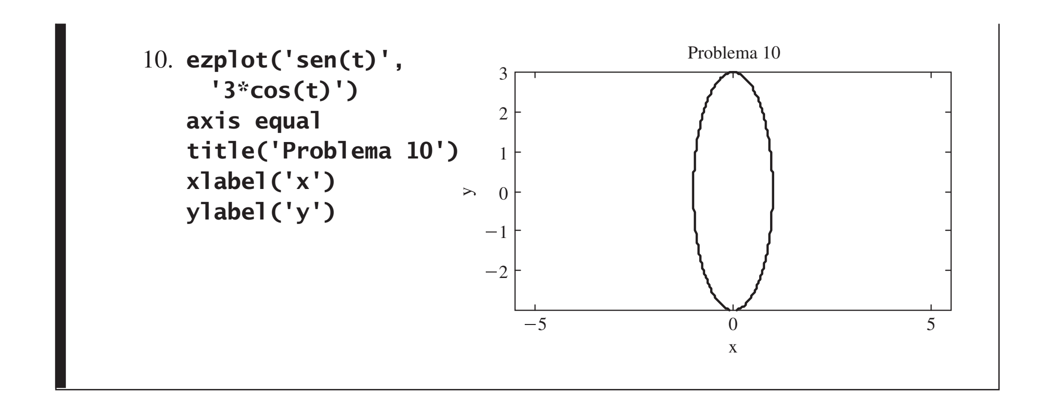
8.
```
ezplot('x^2-y^4=5')
title('Problema 8')
xlabel('x')
ylabel('y')
```



9.
```
ezplot('sen(x)')
hold on
ezplot('cos(x)')
hold off
title('Problema 9')
xlabel('x')
ylabel('y')
```



10.
```
ezplot('sen(t)',
  '3*cos(t)')
axis equal
title('Problema 10')
xlabel('x')
ylabel('y')
```



Ejercicio de práctica 11.7

```
Z=sym('sen(sqrt
   (X^2+Y^2))')
Z =
sin(sqrt(X^2+Y^2))
```

1.
```
ezmesh(Z)
title('Problema 1')
xlabel('x')
ylabel('y')
zlabel('z')
```

Problema 1

2.
```
ezmeshc(Z)
title('Problema 2')
xlabel('x')
ylabel('y')
zlabel('z')
```

Problema 2

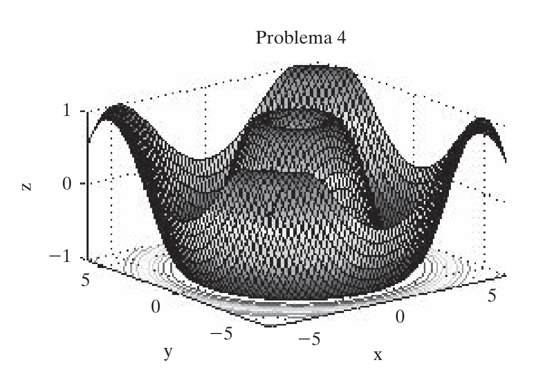
3.
```
ezsurf(Z)
title('Problema 3')
xlabel('x')
ylabel('y')
zlabel('z')
```

Problema 3

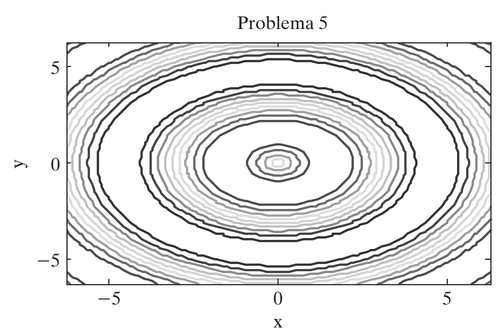
(Continúa)

Ejercicio de práctica 11.7 *(Continuación)*

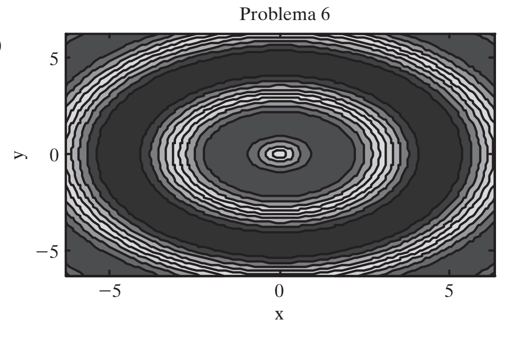
4.
```
ezsurfc(Z)
title('Problema 4')
xlabel('x')
ylabel('y')
zlabel('z')
```



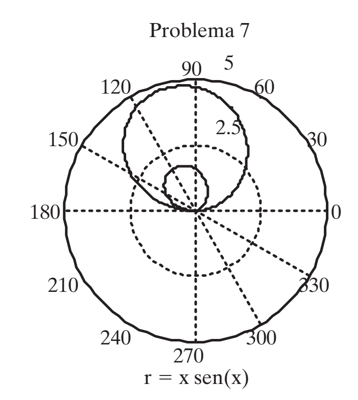
5.
```
ezcontour(Z)
title('Problema 5')
xlabel('x')
ylabel('y')
zlabel('z')
```



6.
```
ezcontourf(Z)
title('Problema 6')
xlabel('x')
ylabel('y')
zlabel('z')
```



7.
```
figure(7)
ezpolar('x*sen(x)')
title('Problema 7')
```



8.
```
t=sym('t');
x = t;
y = sin(t);
z = cos(t);
ezplot3(x,y,z,[0,30])
title('Problema 8')
xlabel('x')
ylabel('y')
zlabel('z')
```

Problema 8

Ejercicio de práctica 11.8

1.
```
diff('x^2+x+1')
ans =
2*x+1
diff('sen(x)')
ans =
cos(x) % o defina x como simbólico
x=sym('x')
x =
x
diff(tan(x))
ans =
1+tan(x)^2
diff(log(x))
ans =
1/x
```

2.
```
diff('a*x^2 + b*x + c')
ans =
2*a*x+b
diff('x^0.5 - 3*y')
ans =
.5/x^.5
diff('tan(x+y)')
ans =
1+tan(x+y)^2
diff('3*x + 4*y - 3*x*y')
ans =
3-3*y
```

3.
```
% Existen varios enfoques diferentes
diff(diff('a*x^2 + b*x + c'))
ans =
2*a
diff('x^0.5 - 3*y',2)
```

(Continúa)

Ejercicio de práctica 11.8 ***(Continuación)***

```
ans =
-.25/x^1.5
diff('tan(x+y)','x',2)
ans =
2*tan(x+y)*(1+tan(x+y)^2)
diff(diff('3*x + 4*y - 3*x*y','x'))
ans =
-3
```

4.
```
diff('y^2-1','y')
ans =
2*y
% o, dado que sólo existe una variable
diff('y^2-1')
ans =
2*y
%
diff('2*y + 3*x^2','y')
ans =
2
diff('a*y + b*x + c*x','y')
ans =
a
```

5.
```
diff('y^2-1','y',2)
ans =
2
% o, dado que sólo existe una variable
diff('y^2-1',2)
ans =
2
%
diff(diff('2*y + 3*x^2','y'),'y')
ans =
0
diff('a*y + b*x + c*x','y',2)
ans =
0
```

Ejercicio de práctica 11.9

1.
```
int('x^2 + x + 1')
ans =
1/3*x^3+1/2*x^2+x
% o defina x como simbólica
x=sym('x')
x =
x
int(x^2 + x + 1)
ans =
1/3*x^3+1/2*x^2+x
int(sin(x))
ans =
-cos(x)
int(tan(x))
ans =
-log(cos(x))
int(log(x))
ans =
x*log(x)-x
```
2.
```
% no necesita especificar que la integración es con
% respecto de x, porque es por defecto
int('a*x^2 + b*x + c')
ans =
1/3*a*x^3+1/2*b*x^2+c*x
int('x^0.5 - 3*y')
ans =
.66666666666666666666666666666667*x^(3/2)-3.*x*y
int('tan(x+y)')
ans =
1/2*log(1+tan(x+y)^2)
int('3*x + 4*y -3*x*y')
ans =
3/2*x^2+4*x*y-3/2*y*x^2
```
3.
```
int(int(x^2 + x + 1))
ans =
1/12*x^4+1/6*x^3+1/2*x^2
int(int(sin(x)))
ans =
-sin(x)
int(int(tan(x)))
ans =
-1/2*i*x^2-x*log(cos(x))+x*log(1+exp(2*i*x))-
  1/2*i*polylog(2,-exp(2*i*x))
int(int(log(x)))
ans =
1/2*x^2*log(x)-3/4*x^2
%
```

(Continúa)

Ejercicio de práctica 11.9 *(Continuación)*

```
int(int('a*x^2 + b*x + c'))
ans =
1/12*a*x^4+1/6*b*x^3+1/2*c*x^2
int(int('x^0.5 - 3*y'))
ans =
.26666666666666666666666666666667*x^(5/2)-
1.5000000000000000000000000000000*y*x^2
int(int('tan(x+y)'))
ans =
-1/4*i*log(tan(x+y)-i)*log(1+tan(x+y)^2)+1/4*i*dilog(-
1/2*i*(tan(x+y)+i))+1/4*i*log(tan(x+y)-i)*log(-
1/2*i*(tan(x+y)+i))+1/8*i*log(tan(x+y)-
i)^2+1/4*i*log(tan(x+y)+i)*log(1+tan(x+y)^2)-
1/4*i*dilog(1/2*i*(tan(x+y)-i))-
1/4*i*log(tan(x+y)+i)*log(1/2*i*(tan(x+y)-i))-
1/8*i*log(tan(x+y)+i)^2
int(int('3*x + 4*y -3*x*y'))
ans =
1/2*x^3+2*y*x^2-1/2*y*x^3
```

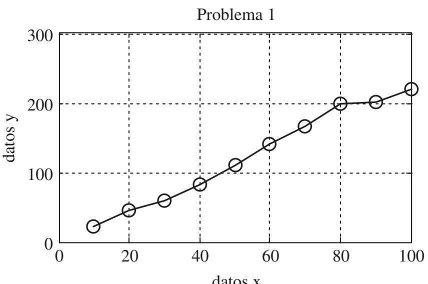
4.
```
int('y^2-1')
ans =
1/3*y^3-y
int('2*y+3*x^2','y')
ans =
y^2+3*y*x^2
int('a*y + b*x + c*z','y')
ans =
1/2*a*y^2+b*x*y+c*z*y
```

5.
```
int(int('y^2-1'))
ans =
1/12*y^4-1/2*y^2
int(int('2*y+3*x^2','y'),'y')
ans =
1/3*y^3+3/2*x^2*y^2
int(int('a*y + b*x + c*z','y'),'y')
ans =
1/6*a*y^3+1/2*b*x*y^2+1/2*c*z*y^2
```

6.
```
int(x^2 + x + 1,0,5)
ans =
355/6
int(sin(x),0,5)
ans =
-cos(5)+1
int(tan(x),0,5)
ans =
NaN
int(log(x),0,5)
ans =
5*log(5)-5
```

Ejercicio de práctica 12.1

1.
```
plot(x,y,'-o')
title('Problema 1')
xlabel('datos x')
ylabel('datos y')
grid on
```



2.
```
interp1(x,y,15)
ans =
   34
```

3.
```
interp1(x,y,15,'spline')
ans =
   35.9547
```

4.
```
interp1(y,x,80)
ans =
   39.0909
```

5.
```
interp1(y,x,80,'spline')
ans =
   39.2238
```

6.
```
new_x=10:2:100;
new_y=interp1(x,y,new_x,'spline');
figure(2)
plot(x,y,'o',new_x,new_y)
legend('datos medidos','interpolación spline')
title('Problema 6')
xlabel('datos x')
ylabel('datos y')
```

Problema 6
datos medidos
interpolación spline
300
200
100
0
datos y
0
20
40
60
80
100
datos x

Ejercicio de práctica 12.2

```
y=10:10:100';
x=[15, 30];
z= [23 33
  45   55
  60   70
  82   92
  111  121
  140  150
  167  177
  198  198
  200  210
  220  230];
```

1. ```
   plot(y,z,'-o')
   title('Problema 1')
   xlabel('datos y')
   ylabel('datos z')
   legend('x=15','x=30')
   ```

2. ```
   new_z=interp2(x,y,z,15,20)
   new_z =
     45
   ```

3. ```
   new_z=interp2(x,y,z,15,20,'spline')
   new_z =
     45
   ```

4. ```
   new_z=interp2(x,y,z,[20,25],y')
   new_z =
    26.3333   29.6667
    48.3333   51.6667
    63.3333   66.6667
    85.3333   88.6667
    114.3333  117.6667
    143.3333  146.6667
    170.3333  173.6667
    198.0000  198.0000
    203.3333  206.6667
    223.3333  226.6667
   ```

Ejercicio de práctica 12.3

```
x=[10:10:100];
y= [23 33
  45   55
  60   70
  82   92
  111  121
  140  150
  167  177
  198  198
  200  210
  220  230]';
```

1.
```
coef=polyfit(x,y(1,:),1)
coef =
  2.3224 -3.1333
```

2.
```
new_x=10:2:100;
new_y=polyval(coef,new_x)
new_y =
 Columns 1 through 6
 20.0909 24.7358 29.3806 34.0255 38.6703 43.3152
 Columns 7 through 12
 47.9600 52.6048 57.2497 61.8945 66.5394 71.1842
 Columns 13 through 18
 75.8291 80.4739 85.1188 89.7636 94.4085 99.0533
 Columns 19 through 24
 103.6982 108.3430 112.9879 117.6327 122.2776 126.9224
 Columns 25 through 30
 131.5673 136.2121 140.8570 145.5018 150.1467 154.7915
 Columns 31 through 36
 159.4364 164.0812 168.7261 173.3709 178.0158 182.6606
 Columns 37 through 42
 187.3055 191.9503 196.5952 201.2400 205.8848 210.5297
 Columns 43 through 46
 215.1745 219.8194 224.4642 229.1091
```

3.
```
figure(1)
plot(x,y(1,:),'o',new_x,new_y)
title('Problema 3 - Modelo de regresión lineal - z = 15')
xlabel('eje x')
ylabel('eje y')
```

(Continúa)

Ejercicio de práctica 12.3 *(Continuación)*

4.
```
figure(2)
coef2=polyfit(x,y(2,:),1)
coef2 =
 2.2921 7.5333
new_y2=polyval(coef2,new_x);
plot(x,y(2,:),'o',new_x,new_y2)
title('Problema 4 - Modelo de regresión lineal -
  z = 30')
xlabel('eje x')
ylabel('eje y')
```


Ejercicio de práctica 12.4

1.
```
x=-5:1:5;
y=x.^3 + 2.*x.^2 - x + 3;
dy_dx=diff(y)./diff(x)
dy_dx =
  42 22 8 0 -2 2 12 28 50 78
dy_dx_analytical=3*x.^2 + 4*x -1
dy_dx_analytical =
  54 31 14 3 -2 -1 6 19 38 63 94
table=[[dy_dx,NaN]',dy_dx_analytical']
table =
   42   54
   22   31
    8   14
    0    3
   -2   -2
    2   -1
   12    6
   28   19
   50   38
   78   63
  NaN   94
% Se agrega NaN al vector dy_dx de modo que la longitud
% de cada vector sería la misma
```

2. a.
```
x=-5:1:5;
y=sin(x);
dy_dx=diff(y)./diff(x);
dy_dx_analytical=cos(x);
table=[[dy_dx,NaN]',dy_dx_analytical']
table =
  -0.2021   0.2837
  -0.8979  -0.6536
  -0.7682  -0.9900
   0.0678  -0.4161
   0.8415   0.5403
   0.8415   1.0000
   0.0678   0.5403
  -0.7682  -0.4161
  -0.8979  -0.9900
  -0.2021  -0.6536
      NaN   0.2837
```

b.
```
x=-5:1:5;
y=x.^5-1;
dy_dx=diff(y)./diff(x);
dy_dx_analytical=5*x.^4;
table=[[dy_dx,NaN]',dy_dx_analytical']
table =
2101  3125
 781  1280
 211  405
  31  80
   1  5
   1  0
  31  5
 211  80
 781  405
2101  1280
 NaN  3125
```

c.
```
x=-5:1:5;
y=5*x.*exp(x);
dy_dx=diff(y)./diff(x);
dy_dx_analytical=5*exp(x) + 5*x.*exp(x);
table=[[dy_dx,NaN]',dy_dx_analytical']
table =
  1.0e+003   *
  -0.0002    -0.0001
  -0.0004    -0.0003
  -0.0006    -0.0005
  -0.0005    -0.0007
  0.0018          0
```

(Continúa)

Ejercicio de práctica 12.4 *(Continuación)*

```
0.0136     0.0050
0.0603     0.0272
0.2274     0.1108
0.7907     0.4017
2.6184     1.3650
   NaN     4.4524
```

Ejercicio de práctica 12.5

1.
```
quad('x.^3+2*x.^2 -x + 3',-1,1)
ans =
  7.3333
quadl('x.^3+2*x.^2 -x + 3',-1,1)
ans =
  7.3333
double(int('x^3+2*x^2 -x + 3',-1,1))
ans =
  7.3333
a=-1;
b=1;
1/4*(b^4-a^4)+2/3*(b^3-a^3)-1/2*(b^2-a^2)+3*(b-a)
ans =
  7.3333
```

2. a.
```
quad('sen(x)',-1,1)
ans =
  0
quadl('sen(x)',-1,1)
ans =
  0
double(int('sen(x)',-1,1))
ans =
  0
a=-1;
b=1;
cos(b)-cos(a)
ans =
  0
```

   b.
```
quad('x.^5-1',-1,1)
ans =
 -2
quadl('x.^5-1',-1,1)
ans =
 -2.0000
double(int('x^5-1',-1,1))
ans =
 -2
```

```
a=-1;
b=1;
(b^6-a^6)/6-(b-a)
ans =
 -2
```

c.
```
quad('5*x.*exp(x)',-1,1)
ans =
 3.6788
quadl('5*x.*exp(x)',-1,1)
ans =
 3.6788
double(int('5*x*exp(x)',-1,1))
ans =
 3.6788
a=-1;
b=1;
-5*(exp(b)-exp(a)) + 5*(b*exp(b)-a*exp(a))
ans =
 3.6788
```

Tabla B.1 Resumen climatológico anual, estación: 310301/13872, Asheville, Carolina del Norte, 1999 (Elev. 2240 pies sobre el nivel del mar; Lat. 35°36′N, Lon. 82°32′W)

Fecha	Temperatura (°F)														Precipitación (pulgadas)									
Elem->	MMXT	MMNT	MNTM	DPNT	HTDD	CLDD	EMXT		EMNP		DT90	DX32	DT32	DT00	TPCP	DPNP	EMXP		TSNW	MXSD		DP01	DP05	DP10
											Número de días				Total	Depart. from Normal	Mayor observado		Nieve, aguanieve			Número de días		
Mes 1999	Máx. Media	Mín. Media	Media	Desv. de la normal	Días grado calenta-miento	Días grado enfria-miento	Más alto	Fecha alto	Más bajo	Fecha bajo	Máx >=90°	Máx <=32°	Mín <=32°	Mín <=0°			Día	Fecha	Caída total	Profun-didad máx.	Fecha máx.	>=.10	>=.50	>=1.0
1	51.4	31.5	41.5	5.8	725	0	78	27	9	5	0	2	16	0	4.56	2.09	1.61	2	2.7	1	31	9	2	2
2	52.6	32.1	42.4	3.5	628	0	66	8	16	14	0	2	16	0	3.07	−0.18	0.79	17	1.2	0T	1	6	3	0
3	52.7	32.5	42.6	−4.8	687	0	76	17	22	8	0	0	19	0	2.47	−1.41	0.62	3	5.3	1	26	8	1	0
4	70.1	48.2	59.2	3.6	197	30	83	10	34	19	0	0	0	0	2.10	−1.02	0.48	27	0.0T	0T	2	6	0	0
5	75.0	51.5	63.3	−0.1	69	25	83	29	40	2	0	0	0	0	2.49	−1.12	0.93	7	0.0	0		5	2	0
6	80.2	60.9	70.6	0.3	4	181	90	8	50	18	1	0	0	0	2.59	−0.68	0.69	29	0.0	0		6	2	0
7	85.7	64.9	75.3	1.6	7	336	96	31	56	13	8	0	0	0	3.87	0.94	0.80	11	0.0	0		10	4	0
8	86.4	63.0	74.7	1.9	0	311	94	13	54	31	7	0	0	0	0.90	−2.86	0.29	8	0.0	0		4	0	0
9	79.1	54.6	66.9	0.2	43	106	91	2	39	23	3	0	0	0	1.72	−1.48	0.75	28	0.0	0		4	1	0
10	67.6	45.5	56.6	0.4	255	1	78	15	28	25	0	0	2	0	1.53	−1.24	0.59	4	0.0	0		3	2	0
11	62.2	40.7	51.5	4.0	397	0	76	9	26	30	0	0	8	0	3.48	0.56	1.71	25	0.3	0		5	3	1
12	53.6	30.5	42.1	2.7	706	0	69	4	15	25	0	0	20	0	1.07	−1.72	0.65	13	0.0T	0T	17	3	1	0
Anual	68.0	46.3	57.2	1.6	3718	990	96	Jul	9	Ene	19	4	81	0	29.85	−8.12	1.71	Nov	9.5	1	Mar	69	21	3

Notas

(blanco) No reportado.

- \+ Ocurrió en una o más fechas previas durante el mes. La fecha en el campo Fecha es el último día de ocurrencia. Usado sólo hasta diciembre de 1983.
- A Cantidad acumulada. Este valor es un total que puede incluir datos de un mes o meses o año (para valor anual) previos.
- B Total ajustado. Valores totales mensuales con base en datos proporcionales disponibles a través de todo el mes.
- E Total anual o mensual estimado.
- X Medias o totales mensuales con base en series de tiempo incompletas. Faltan 1 a 9 días. Las medias o totales anuales incluyen uno o más meses que tienen de 1 a 9 días faltantes.
- M Se usa para indicar elemento de datos faltante.
- T Trazo de precipitación, nevada o nieve profunda. El valor de datos de precipitación será = cero.
- S Cantidad de precipitación en continua acumulación. El total se incluirá en un valor mensual o anual subsecuente. Ejemplo: días 1-20 tienen 1.35 pulgadas de precipitación, luego comienza un periodo de acumulación. El elemento TPCP sería entonces 001355 y el valor de cantidad total acumulada aparece en un valor mensual subsecuente. Si TPCP = "M" no hubo precipitación medida durante el mes. La bandera se pone en "S" y la cantidad total acumulada aparece en un valor mensual subsecuente.

U.S. Department of Commerce National Oceanic & Atmospheric Administration

Índice analítico

D